

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



THESIS

AUTONOMOUS AGENTS FOR DIGITAL NETWORK MAXIMIZATION

by

Michael W. DaBose

September 1997

Thesis Advisor:

Luqi

Co-Advisor:

V. Berzins

Approved for public release; distribution is unlimited

DTIC QUALITY INSPECTED 3

19980223 126

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302 and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)		REPORT DATE September 1997	REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE AUTONOMOUS AGENTS FOR DIGITAL NETWORK MAXIMIZATION			5. FUNDING NUMBERS	
AUTHOR DaBose, Michael W.				
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSOR/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Science Assistance Program			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense of the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
ABSTRACT (Maximum 200 words) <p>The advent of the computer age has brought about a plenitude of benefits to the human race. Included with these benefits has been the ever increasing demand to transfer exponentially increasing amounts of information, and the associated problems of information sharing. The focus of this thesis, Naval Science Assistance Program (NSAP), and Office of Naval Research (ONR) funded research effort, has been to best utilize available digital communications assets in the radio frequency (RF) spectrum to allow sufficient transfer of information providing DOD assets flexible, rapid, and in-flight reprogramming, re-planning of strike and cruise missile assets, to engage a high value, emergent target, in the shortest possible time. The postulated methods of utilizing autonomous agents to manage information flow across network nodes has applicability to all digital networks.</p> <p>Based upon the pioneering work of Pattie Maes, at Massachusetts Institute of Technology (MIT), and previous examination of communications node management, the implementation of independent processes, working on behalf of a host system, to optimize the effective meaningful throughput on a communications channel is not only desirable, but necessary. The evolution of semi intelligent software, whether called Artificial Intelligence, Intelligent Agents, or Autonomous Agents, has reached a level of sophistication allowing the insertion of meaningful articulated processes within existing, and future systems to maximize the network efficiency systematically. Recent work by Michael Cohen on Sodabots, and the evolution of user interactive TinyMUDS of the Maas-Neotek family, a virtual type personality environment, has demonstrated the ability of software to deal with dynamic and changing conditions. The exponential increase in micro-processor power has, for the first time, made available the hardware for such agent implementations as compact, self contained, embedded systems, in direct support of larger existing systems.</p>				
14. SUBJECT TERMS agents, networks, digital, communications, maximization, AI, Software Engineering			15. NUMBER OF PAGES 188	
			16. PRICE CODE	
SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18

Approved for public release; distribution is unlimited

AUTONOMOUS AGENTS FOR DIGITAL NETWORK MAXIMIZATION

Michael W. DaBose
B.A., Miami University, Oxford, Ohio, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

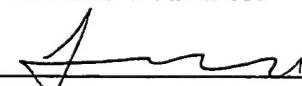
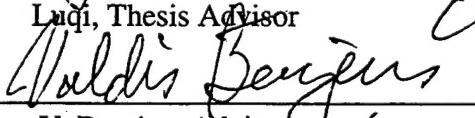
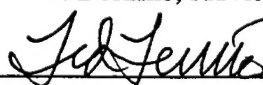
**NAVAL POSTGRADUATE SCHOOL
September 1997**

Author:



Michael W. DaBose

Approved by:


Luigi, Thesis Advisor
V. Berzins, Advisor
T. Lewis, Chairman, Department of Computer Science

ABSTRACT

An important problem arising from the increased sharing of information across networks is bandwidth constraint. Then limitations of communications channels in the transmission of volumous information is the singular bottleneck dictating processing capability and robustness of current and future distributed systems. Bandwidth utilization with the goal of optimizing the actual information transmitted, has to date, been ignored. Many of the current network strategies, both commercial, and tactical, rely on the repeated broadcast of a standardized message. As a result, much available bandwidth is wasted. The specific approach taken to maximize specific network node throughput on a digital network is a three-layer paradigm, managed by an embedded autonomous software agent located at each network node. The first layer consists of a network specific strategy for reducing the message content. The second layer is a frame by frame analysis of the reduced message content, to determine the best compression method to be applied to the information itself (MPEG, etc.). Finally a packaging strategy is utilized to maximize the compressed content for each specific network packet. The first phase of a proof of concept prototype has been implemented. Initial results, via a network simulation, have demonstrated a quantative 300% plus increase in effective information throughput capability, utilizing the same bandwidth. Since this approach is an embedded technique, existing network hardware, software, and standards remain uneffected. A side benefit witnessed is increased network responsiveness, due to increased information flow in a timely manner. In terms of processing time required, the cost is more than compensated for by increased network efficiency. The net result is a more efficient and responsive network capability. Future efforts will implement the entire node management capability described in this thesis. It is anticipated this capability will be introduced to the opporational fleet within the next five to seven years.

TABLE OF CONTENTS

I.	INTRODUCTION	1
	A. GENERAL	1
	B. PROBLEM STATEMENT	2
II.	RISK ASSESSMENT FOUNDATION FOR DESIGN PRINCIPLES	5
	A. INTRODUCTION	5
	B. PROBLEM STATEMENT	6
	C. GLOBAL REAL TIME RETARGETING REQUIREMENT	7
	D. SCIENCE AND TECHNOLOGY SHORTFALL	8
	E. TECHNICAL APPROACH	9
	1. General Discussion	9
	F. ASSUMPTIONS	10
	G. AGENTS FOR TRANSPARENT OPTIMIZATION OF INFORMATION TRANSMISSION	10
	H. LINK 16 GENERAL DISCRIPTION	12
	1. Link-16 Functional Description	12
	2. Link-16 Operation	12
	3. Link-16 Configuration	13
	I. AGENT-BASED TECHNIQUES FOR REAL-TIME SYSTEMS	14
	1. Introduction	14
	2. Intelligent Real-Time Systems (Robotics)	15
	3. Real-Time Expert Systems	19
	J. OTHER CRITICAL TECHNOLOGIES	21
	1. Information Theory: Mathematics of Communications	21
III.	ENGINEERING DESIGN FOR A NODE AGENT	29
	A. CODESIGN: SOFTWARE ENGINEERING FOR REAL-TIME SYSTEMS	29
	B. AN AGENT FOR LINK 16/22	32
	1. A Concept	32
	C. SOFTWARE REQUIREMENTS - A FUNCTIONAL DISECTION LEADING TO A PROPOSED DESIGN FOR IMPLEMENTATION	33
	1. Functions & Operations	34
	D. FUNCTIONAL REQUIREMENTS	35
	E. KNOWLEDGE-BASED COMPONENTS	37
	1. General	37
	2. Semantic Networks	37
	F. FACTS	40
	1. General Forms	40
	2. FactSets	41
	G. PRODUCTION RULES	41
	1. General Forms-Antecedents and Consequents	41
	2. Comparisons	41

3. Antecedent format	42
4. Consequents	43
5. Sample Complete Rule	43
H. CONCATENATION IN THE RULES	44
I. ANY/THAT/THOSE CONSTRUCTIONS	46
1. General	46
2. Subsets	46
3. Evaluation Order	47
J. DIRECTIVES	48
1. DEINstantiate	48
2. LOCK/UNLOCK	48
3. NUMBEROF	49
4. NAMEOF	49
5. INFER	50
6. PARALLEL	50
K. SPECIAL TERMS	50
1. NULL	51
2. MINUS/PLUS/DIVIDED_BY (Slot Arithmetic)	51
3. FIELD	51
4. NCHARS	52
L. VALUE ARITHMETIC	53
M. CONCATENATION IN NAMES	54
N. VARIABLES, POINTERS, AND INDIRECTION	54
O. REASONING WITH UNCERTAINTY	55
P. THE AGENT ENGINE	55
1. Rule Order and the Academic Paradigm	56
2. Automatic Rule Sorting and Parallel Processing	57
3. Functions within Consequents	57
4. Nested Inferencing	58
5. Compiling	58
6. The WHILE Construction	59
7. BREAK	59
Q. SHARED MEMORY	59
R. META CONTROL AND META LANGUAGE	60
1. Meta Control Functions	61
IV. AGENT ARCHITECTURE	65
A. LINK MANAGER	65
B. SIMPLIFYING ASSUMPTIONS FOR THE PROTOTYPE	67
1. Left Input Process	68
2. Left Output Process	71
3. Exception Handler	72
C. DELTA COMPONENTS	83
1. Encoder	84

2. Decoder	87
D. SHARED COMPONENTS	90
V. AGENT METHODS FOR LINK 16	92
A. INTRODUCTION	92
B. ATOMIC DATA ELEMENT TRANSMISSION ("DELTA MESSAGES")	92
C. UPDATE BUNDLING	97
D. EXTRAPOLATION-DRIVEN UPDATES	98
E. TRADITIONAL COMPRESSION	101
F. EXTENSIONS	101
VI. SUMMARY	103
A. WORK ACCOMPLISHMENTS	103
B. WORK TO BE DONE IN SUBSEQUENT PHASES	103
VII. FUTURE WORK	105
LIST OF REFERENCES	108
APPENDIX	113
INITIAL DISTRIBUTION LIST	171

LIST OF ACRONYMS

agent	in the context of this thesis, a goal seeking, semi-intelligent, independent program working on behalf of a host system to increase efficiency
RTR	real time retargeting
C2P	command control processor
JTIDS	Joint Tactical Information Distribution System, the system name for a spread spectrum, anti-jam, means of time synchronized, digital communication
CDLMS	Common Datalink Management System
RF	radio frequency, as in RF transmissions
link	herein refers to military specific digital communications networks operating over a radio frequency spectrum (RF)

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professors Luqi, Berzins, and the entire Computer Science Department staff for their guidance, support, and patience. Without their teachings this work would not have been possible.

A special thanks to Naval Research, Development, Test & Evaluation (NRaD), San Diego, specifically Mr. Rod Smith, and John Iaia, whose support for my acceptance into this masters program, made this valuable effort possible. Thanks to the Office of Naval Research (ONR), Naval Science Assistance Program (NSAP) for the funding necessary to conduct this initial research.

I would like to acknowledge the support and insight of Lonnie Nessler and Jason Durham, who labored long hours with me on the initial concepts formulating this research project.

Finally, a fond thank you to my father, Walter DaBose, who encouraged me to pursue this advanced degree program. He will not be forgotten . . .

I. INTRODUCTION

A. GENERAL

The advent of the computer age has brought about a plenitude of benefits to the human race. Included with these benefits has been the ever increasing demand to transfer exponentially increasing amounts of information, and the associated problems of information sharing. The focus of this thesis, associated Naval Science Assistance Program (NSAP), and Office of Naval Research (ONR) funded research effort, has been to best utilize available digital communications assets in the radio frequency (RF) spectrum to allow sufficient transfer of information providing DOD assets flexible, rapid, and in-flight reprogramming, re-planning of strike and cruise missile assets, to engage a high value, emergent target, in the shortest possible time. The postulated methods of utilizing autonomous agents to manage information flow across network nodes has applicability to all digital networks.

Based upon the pioneering work of Pattie Maes, at Massachusetts Institute of Technology (MIT) (4, 17, 31, 33, 53, 54), and previous examination of communications node management (37, 38), the implementation of independent processes, working on behalf of a host system, to optimize the effective meaningful throughput on a communications channel is not only desirable, but necessary. The evolution of semi intelligent software, whether called Artificial Intelligence, Intelligent Agents, or Autonomous Agents, has reached a level of sophistication allowing the insertion of meaningful articulated processes within existing, and future systems maximize the network efficiency systematically. Recent work by Michael Cohen on Sodabots (8), and the evolution of user interactive TinyMUDs (Multiple User Dimension) of the Maas-Neotek family (4), a virtual type personality environment, has demonstrated the ability of software to deal with dynamic and changing conditions. The additional, and exponential increase in micro-processor power has, for the first time, made available the hardware for such agent implementations as compact, self contained, embedded systems, in direct support of larger existing systems.

B. PROBLEM STATEMENT

At present, tactical command and control networks, such as Link 16, do not employ data management techniques to maximize the utilization of the RF spectrum. With respect to the surveillance Net Participation Group (NPG), information updates on platform tracks are transmitted in full and at a high rate. This is true even when only a single data field (e.g., the location) of the track information has changed. The rapid repetition of redundant information is a characteristic of other NPGs as well. One of the reasons for this repetition is that a listener, if he misses a message, can simply ignore it—another chance to receive the information will soon follow. An important drawback of this passive design, however, is that the bandwidth needed to support RTR requirements will not be available.

It is assumed that no new tactical networks (hereinafter referred to as “links”) will be available for the foreseeable future, and certainly not in time to meet the needs of the RTR Cruise Missile demonstration and the RTR program that supports it.

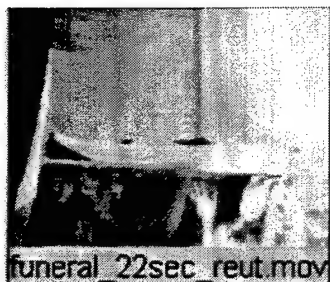
Therefore every effort must be made to obtain more efficient use of bandwidth from existing links. There are five methods, independent of efforts to handle network entry, and passive synchronization, that this effort will explore: Atomic Data Element Transmission, Update Bundling, Extrapolation-Driven Updates, Traditional Compression, and Active Network Management. In FY96, message size reduction was demonstrated on a closed loop link simulation, built upon existing assets. These initial achievement are leading toward benchmark demonstrations utilizing NRD's Systems Integration Facility (SIF) to establish best-case effectiveness. In later years, benchmarks will be established for additional techniques.

The ultimate aim is to increase the available bandwidth, through efficient and more formal management of existing tactical networks, without changing network operation rules or message integrity assurance. This capability will result in a new way of handling information on existing links, with the possibility of extending capability to a new message specification while coexisting within the given operational network.

As demonstrated by the news presented every day, reactionism is ever increasing in today's complicated world. The demise of the former Soviet Union has eased the potential for

armed exchanged by responsible parties, but has dramatically increased the potential of hostilities with the slightest trigger event leading to potential larger exchanges, and loss of life.

A small trigger event



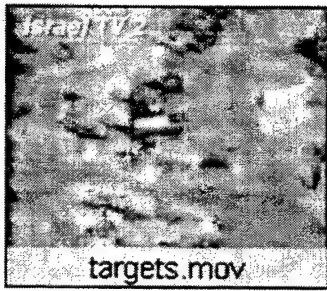
can lead to aggression



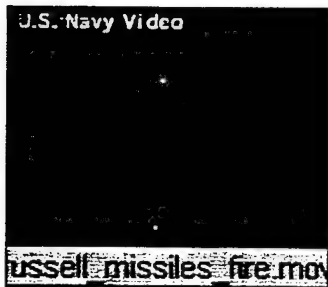
leading to a response



leading to escalation



leading to a larger response



all within a very short period of time. One key point to bear in mind, even the smallest have not organizations, or countries have access to weapons of mass destruction. The rational mindset does not necessarily apply here, therefor the need for rapid response dictating efficient communications throughput capacity.

II. RISK ASSESSMENT FOUNDATION FOR DESIGN PRINCIPLES

A. INTRODUCTION

This document describes the engineering design of an autonomous or knowledge-based agent designed to increase effective throughput on tactical links. The agent itself is called "Agent," and each instance of the agent is identical on all nodes in the network. The sole assumption made by the architecture is that each node on the network will have an agent associated with it able to intercept and manipulate both incoming and outgoing messages.

The architecture is knowledge-based in all other respects. Thus it is not specific to any particular tactical link or digital network. To operate on some other network requires modifications to the knowledge bases and facts, but no modifications to the agent itself.

This document is not intended to function as a tutorial for knowledge-based systems concepts. It assumes a high level of sophistication in traditional concepts and addresses where the engineering design departs from them. Neither is this document intended to be a tutorial about Link-16 or other tactical communications networks.

The principal accomplishment was the design of an inference engine specifically tailored for intelligent agents. The special features of this inference engine are described in depth so that detailed designers can know what to do.

Each process in the system is controlled by meta rules and a special meta controller that is part of the agent run-time environment. Although an elaborate meta language was designed, adding sophistication to the inference engine reduced reliance on sophisticated meta rules. Very few functions are involved at present.

The run-time environment for the agent was not designed. The assumption was that one or more commercial agent-building tool kits, such as IBM's ABE (Agent Building Environment) would be available to initialize and run the agent. Since these tool kits can be modified to use a custom inference engine, and because the facts and rules used by the Agent system can in general be translated to and from KIF, the assumption that a tool kit would be available at the time of detailed design seems reasonable to the author. However, there is nothing in this system that requires the use of such a tool kit. The run-time mechanisms necessary to control the

processes and inferencing are not complicated. This supports an implementation structure in which the agent can run on its own machine inserted at each node between the node and the network.

This document assumes Link-16 concepts but the methods of autonomous agents are not specific to Link-16.

B. PROBLEM STATEMENT

At present, tactical command and control networks, such as Link 16, do not employ data management techniques to maximize the utilization of the RF spectrum. With respect to the surveillance Net Participation Group (NPG), information updates on platform tracks are transmitted in full and at a high rate. This is true even when only a single data field (e.g., the location) of the track information has changed. The rapid repetition of redundant information is a characteristic of other NPGs as well. One of the reasons for this repetition is that a listener, if he misses a message, can simply ignore it—another chance to receive the information will soon follow. An important drawback of this passive design, however, is that the bandwidth needed to support information exchange requirements will not be available.

It is assumed that no new tactical networks (hereinafter referred to as “links”) will be available for the foreseeable future.

Therefore every effort must be made to obtain more efficient use of bandwidth from existing links. There are five methods, independent of efforts to handle network entry, and passive synchronization, that this effort will explore: Atomic Data Element Transmission, Update Bundling, Extrapolation-Driven Updates, Traditional Compression, and Active Network Management. In FY96, message size reduction was demonstrated on a closed loop link simulation, built upon existing assets. These initial achievements are leading toward benchmark demonstrations utilizing Naval Research Development Test & Evaluation (NRaD) Systems Integration Facility (SIF) to establish best-case effectiveness. In later years, benchmarks will be established for additional techniques.

The ultimate aim is to increase the available bandwidth, through efficient and more formal management of existing tactical networks, without changing network operation rules or

message integrity assurance. This capability will result in a new way of handling information on existing links, with the possibility of extending capability to a new message specification while coexisting within the given operational network.

C. GLOBAL REAL TIME RETARGETING REQUIREMENT

The basis for this thesis was guided by sponsor desires to support rapid mission re-allocation of assets in support of emergent threats on a real time basis. FY96 Program Execution Plan for Real-Time Retargeting (RTR) provides a good list of the general requirements for real-time retargeting, and basis by which the concept and development of agents is being accomplished:

The ONR Real-Time Retargeting (RTR) Accelerated Capability Initiative (ACI) is aimed at providing the cruise missile and tactical aircraft warfighters a new capability to redirect their strike forces in real-time to respond to current dynamics of the battlefield. The focus of the ACI is on time-critical targets. To engage such targets, these warfighters must plan/replan, control, and execute strikes in 5 to 30 minutes, instead of the military's current operational capability of many hours. Current shortfalls that inhibit these warfighters from responding in real-time are: (1) lack of automated real-time mission (re)planning capability, (2) lack of communications availability and capacity, (3) lack of automated target detection and recognition, (4) lack of rapid mission management and execution decision aids, (5) lack of real-time terminal targeting information dissemination between surveillance assets and strike platforms, and potentially between the strike platforms themselves, and (6) lack of necessary available bandwidth to support robust rapid data transmission in support of RTR. This equates into a lack of real-time threat awareness on board the strike platform.

Shortfalls two and six are addressed by this effort. Without requiring a change to the net protocols or integrity assurance, this effort is developing methods for extending the effective bandwidth of existing tactical networks. By utilizing algorithms and system structures which are typically not associated with throughput optimization, substantial improvements can be realized for extending the communications capabilities of existing networks. The technologies developed by this research will support the capabilities of Tomahawk Block IV+ or TSTAR. It will also apply to any digital network through optimization of available bandwidth. This effort is not a cure-all for network bottlenecks, but rather a best use of what is available.

D. SCIENCE AND TECHNOLOGY SHORTFALL

Link 16 is predominately a passive system. This means that messages, such as track updates, are transmitted so frequently that they need not be acknowledged by receiving platforms. Therefore, if a platform misses a message, it need only wait a few seconds for an update. The goal of the design within the surveillance NPG is to provide a complete tactical picture in twelve seconds or less, for as many as two thousand tracks. Although the total load on the link by the surveillance NPG is not as great as that produced by voice communications or video conferencing, it is significant nevertheless.

In order to adapt the link to RTR requirements, both transmitting and receiving platforms need to be smarter, reducing both the level of redundancy in the messages and the frequency with which they are transmitted. The idea is to insert processor capability between the C2P and the JTIDS terminal on all platforms to distribute network control and manage message content in a smarter way.¹ Because of the complexity of the rules of the network and the communications between C2P and JTIDS, an agent based architecture is the ideal candidate: it is designed to operate transparently, can take action on its own volition, is extensible and flexible, and is capable of being included, and / or integrated, in future system upgrades, and will handle the complexity of network entry / passive synchronization, while maintaining maximized throughput capability of only "changed" information. Distributed control architectures in an object-oriented environment can produce smart and complex behaviors that are not generally achievable with traditional techniques.

¹ The eventual target for this work is probably the CDLMS effort (Common Data Link Management System), which will at some point produce a system replacing the current C2P.

E. TECHNICAL APPROACH

1. General Discussion

Depending upon available resources, the approach to the FY96 tasks was to establish performance objectives and baselines. Sanitized data from actual network operations were acquired from the Systems Integration Facility (SIF). This data established baseline for evaluating redundancies in network communications. From this data, a decrease in loading is expected to result from the technique(s) being developed by this effort. Additional candidate techniques are under development and will be tested separately.

The software written to date has employed reusable, object-oriented methods/techniques. A need for a real-time object-oriented data base management system (OODBMS) is anticipated as system capabilities are developed. The system architecture is designed to incorporate such tools as they are needed and can be exploited. Also, a PMW 159 sponsored effort called the Common Data Link Management System (CDLMS), will result in an object-oriented tactical communication system. Thus, employing an object oriented style will reduce costs for future integration efforts.

A number of commercial OODBMS's, such as ObjectStore, Versant, ONTOS, and Polyhedra, are under consideration and are to be evaluated as the need arises. Their ability to satisfy real-time demands of link messaging, is considered a critical feature. The ability to manage storage and retrieval of temporal data, is also a critical factor.

Finally, objects are by definition, independently executing processes that are designed for maintainability and reuse. From their hierarchical orientation, the levels of abstraction in object oriented systems are more explicit and straightforward than traditional styles. As the initial exploratory efforts are performed, and lower level object methods are created, developing more abstract levels of processing is relatively fast, reliable, and adaptable toward new capabilities.

F. ASSUMPTIONS

The following assumptions are made with respect to the proposed modifications suggested herein:

- Any modifications made between two C2P / JTIDS terminal linkages can be accomplished, as long as the rules for the particular network in question are followed.

- Messages will have specific data elements recognizable to the C2P. As for transmission methods are concerned, data is data, and can be manipulated as desired, so long as the reconstituted message elements follow the assumption above and do not violate any link integrity requirements.

- Link rules enforce standards. As long as the end results obey link rules, the proposed modifications can be performed, given timeliness and robustness are maintained.

- Rules evolve as systems evolve, tactics change, and lessons are learned. Software designed under this effort is flexible, allowing changes in rules without forcing system redesign.

G. AGENTS FOR TRANSPARENT OPTIMIZATION OF INFORMATION TRANSMISSION

An "agent" is a relatively-independent software entity, or collection of such entities, that performs tasks on behalf of human users but also can perform on behalf of other computer applications or systems. There are two important features of software agents. First, they are anthropomorphic processes. As much as possible, agents behave and communicate in ways that are similar to and compatible with human behavior and communication. As such, they are not necessarily "smart" or "intelligent," but like humans, they tend to be goal driven.

Resource management is a common task for software agents. In this capacity, they operate at a higher level of abstraction than the user or system they oversee. Often they are separate processes initiated prior to the initiation of the system they oversee. Like human managers, they are able to "watch" user or system operations, detect problems, formulate corrective plans, and carry them out. Some can study the effects of their interventions and modify their behavior accordingly.

Distributed set of intelligent agents may provide the means and method for overseeing the operations of an operational network such as Link-16. Present at each platform, observing transmissions, received messages, and user interactions, agents can be designed to optimize link performance and enhance operator efficiency.

In late 1996, a demonstration capability was developed and an initial evaluation of network messages was performed. In 1997, an initial design of a software agent, hereinafter called "Agent," was prototyped (proof of concept), while longer-range architectural ideas are considered. In the out years, the final agent architecture will:

- Accommodate the differing needs of different platforms without recoding;
- Understand the rules of the links so that all its actions will conform to them;
- Be extensible, either directly or through cloning with different knowledge bases, to other messages and functions;

When a message is sent from the C2P to the JTIDS, the agent determines, by checking its type, whether or not it is of interest. If it is not, the message is simply reasserted on the outgoing bus.

The design goal for Agent architectures is to be able to accommodate future functional requirements by adding additional knowledge, leaving the engines and interpreters largely intact. The result is a system that can be extended at minimal cost.

The key design issue is real-time performance. In general, the higher the level of abstraction, the higher the overhead. Therefore, particular implementations, such as Link 16, are driven by the time constraints of the links, available system resources, and scalability of the agent-based architecture. Fortunately, agent-based architectures for Intelligent Real-Time Systems and Real-Time Expert Systems, have addressed similar hard real-time constraints. This work draws upon those results.

H. LINK 16 GENERAL DESCRIPTION

1. Link-16 Functional Description

Link-16, or TADIL J, uses the Joint Tactical Information Distribution System. JTIDS refers to the communication component of Link-16 that encompasses the software, hardware, RF equipment and the waveform that they generate. Among NATO subscribers, the equivalent term for JTIDS is the Multifunctional Information Distribution System (MIDS). Link-16 employs netted communication techniques and a standard message format for exchanging digital information among airborne, land-based and shipboard tactical data systems. Link-16 does not significantly change the basic concepts of tactical data link information exchange supported for many years by Link-11 and Link-4A. Link-16 provides technical and operational improvements to existing tactical data link capabilities. The improvements include nodelessness; electronic countermeasure (ECM) resistance; flexibility of communication operations; separate transmission and data security; increased number of participants; increased data capacity; network navigation features; and secure voice. Link-16 also uses a Time Division Multiple Access (TDMA) architecture, which provides multiple and simultaneous communication nets.

2. Link-16 Operation

Link-16 operates at L-band frequencies (969 - 1206 MHz) with 3 MHz channel spacing and a power output of 200 - 1260. It transmits a frequency-hopping transmission pattern using 51 frequencies at 3 MHz intervals, excluding identification friend or foe (IFF) sub-bands. Frequency-hopping and other spread-spectrum techniques make Link-16 resistant to jamming and data encryption makes it secure. The Link-16 message standard consists of one or more 70-bit words plus, error detection, correction bits, and symbols. The word formats are used to allow a single message to convey position, track data, weapons control and command messages.

3. Link-16 Configuration

The major components of the Navy shipboard Link-16 system include the following: Tactical Data System (TDS), Command and Control Processor (C2P), JTIDS terminal and JTIDS antennas. The TDS and C2P provide the tactical data to be exchanged. The JTIDS terminal and antennas provide the secure, anti-jam, increased capacity waveform. The JTIDS terminal is composed of two major components: the receiver/transmitter (R/T) and the data processor group (DPG). The R/T is common to all platforms. The DPG contains a digital data processor and an interface unit (IU). The IU is tailored specifically to each type of platform.

There are two configurations of Link-16, known as Model 4 and Model 5. The Model 4 implementation of Link-16-also referred to as Block 0 on Advanced Combat Direction System (ACDS) platforms-was designed as a transparent equipment upgrade to existing ship's tactical systems. Model 5-also referred to as Block 1 on ACDS platforms-is the full and complete implementation of Link-16 in accordance with OS5161.

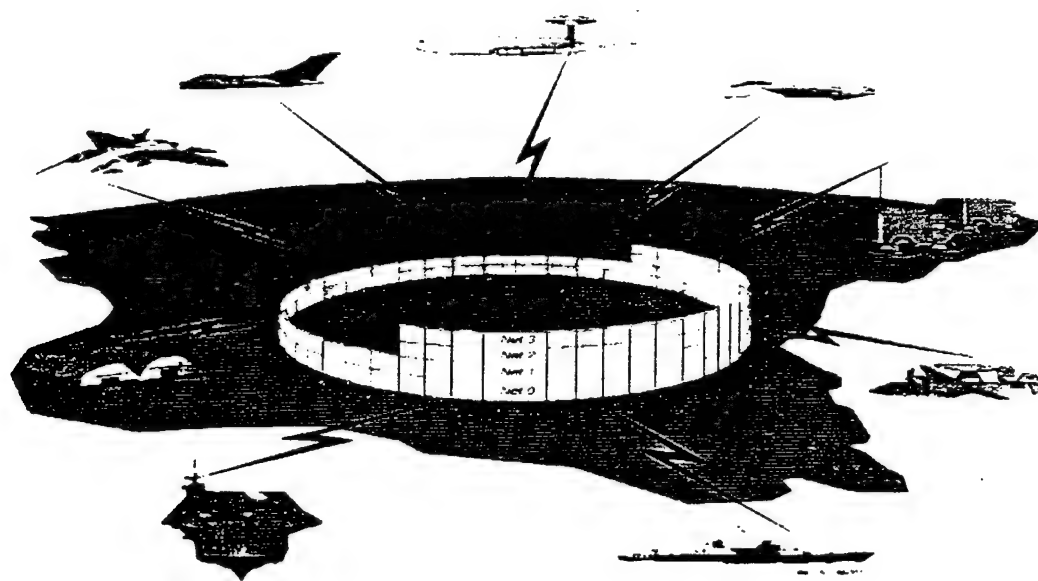


Figure 1

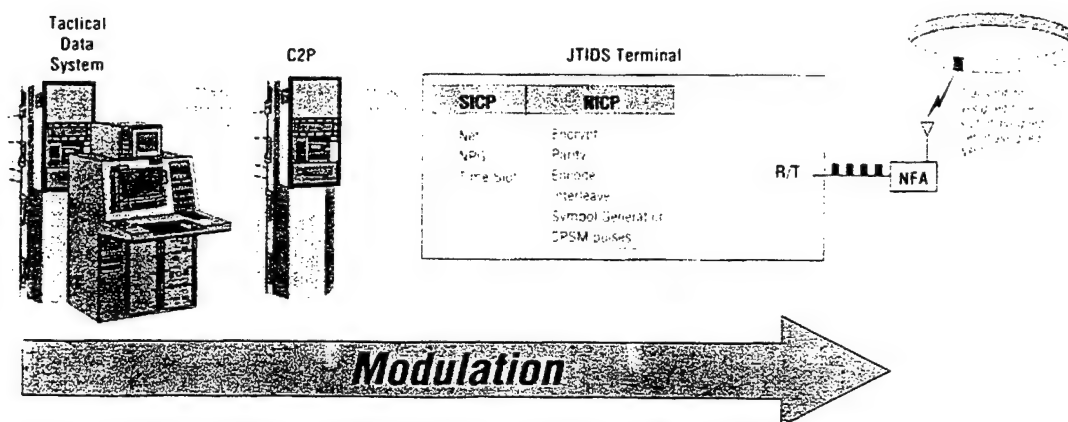


Figure 2

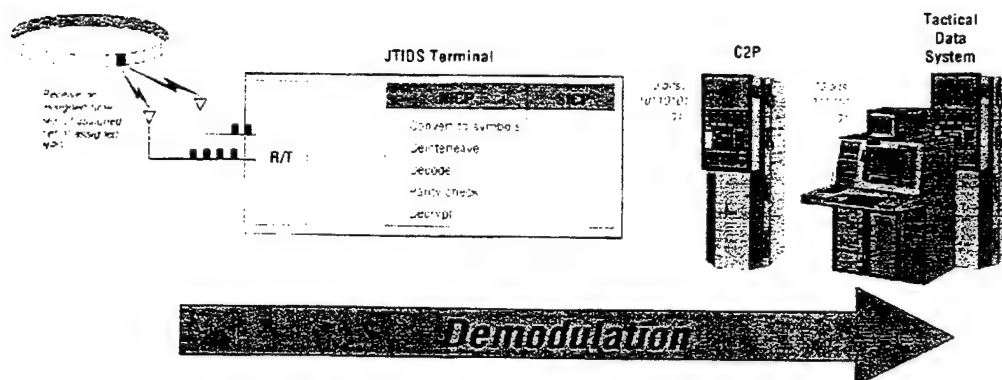


Figure 3

I. AGENT-BASED TECHNIQUES FOR REAL-TIME SYSTEMS

1. Introduction

The following summary is from Huang [63]:

George Saradis introduces a three-level hierarchy [Saradis, 1985]. Acar and Ozguner present an alternative architecture that organizes the system into a multiresolution hierarchy by identifying its components and examining the physical relationships among them [Acar and Ozguner, 1990]. Antsaklis and Passino describe a hierarchical control architecture that uses a hybrid approach to model systems with a high degree of

autonomy [Antsaklis and Passino, 1993]. Successive delegation of duties from higher to lower levels is one of this hierarchy's important characteristics. In another chapter of Antsaklis and Passino, Meystel describes a nested hierarchical control theory that includes the concept of treating design and control as a continuum [Meystel, 1993].

Regarding implementation of intelligent control systems, research has focused on software technologies and computer-aided software-engineering environments. Sweet and his colleagues identify key software technologies for the Aerospace Industries Association [Sweet et al, 1989]. Simmons describes a Task Control Architecture [Simmons 1990]. One limitation of TCA might be scalability because it is not intended to model multiple-cooperating agents. Object-oriented paradigms are becoming popular for handling the representation problems of software systems, but they are not suitable for all problems. For example, Schneider and his colleagues developed a flexible, object-oriented, real-time software implementation tool called ControlShell [Schneider et al., 1994]. But this tool does not address the issue of architecture; perhaps a reference-model architecture could complement its capabilities.

Huang is at the National Institute of Standards and Technology (NIST) where, for over two decades, Jim Albus has been developing the Real-Time Control System (RCS) reference-model architecture. A reference-model does not describe how system structures are represented and implemented, but rather seek to conceptualize the high level functionality and process interdependency. Object-oriented (OO) paradigms have become popular because they can reflect the inherent structure of reference models. Given a reference model or system architecture, the actual implementation, as driven by the model or architecture, is typically a separate design effort. Intelligent real-time system architectures are discussed in more detail in the next section.

2. Intelligent Real-Time Systems (Robotics)

An approach and methodology for engineering intelligent real-time systems is discussed by Durham [44, 45, 46]. In terms of systems architectures, Durham considers intelligent real-time systems functionally equivalent to intelligent robot systems. The two types of systems share similar requirements and objectives, and only differ in terms of the "plant" or environment for which they interact. When a robotic agent interacts with a subset of a system such that its "plant" is restricted to system software, such agents have been called "softbots," i.e. software robots. Softbot applications do not typically have hard real-time requirements and the real-time

limitations of softbots is an open issue. To address such issues, this effort capitalizes on the recent advances in autonomous systems and telerobotics.

Figure 4, below is a diagram of a generic intelligent system architecture derived from some early results in the ARPA Autonomous Land Vehicle (ALV) program. Figure 5 is from Meystel's recent work in "Semiotic" system architectures [59]. These diagrams illustrate the generic high level architecture of real-time intelligent systems.

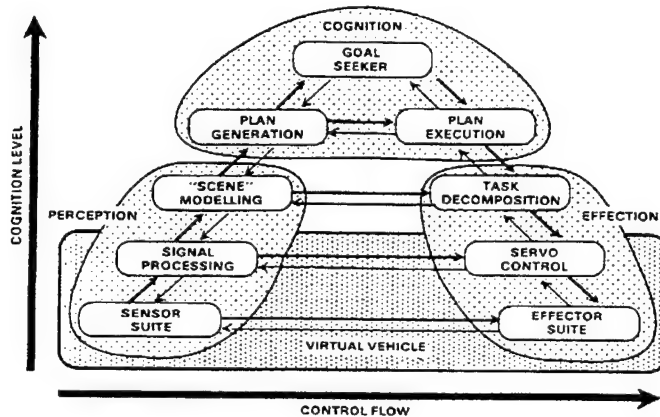


Figure 4

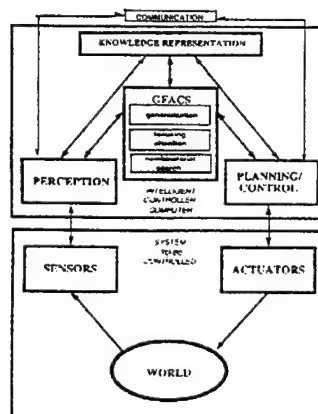


Figure 5

Figures 6 and 7 are two diagrams from an intelligent system architecture as described by Valavanis and Saridis [60]. These diagrams further illustrate a functional organization and

layering. The highest level processes are similar to the executive level within a business organization. Longer term organizational structures, such as strategies and mission statements, are the types of objects and data structures to be processed. The coordination of day-to-day operations is a lower level management function. Specification and design of specific tasks is a critical "coordinator" function. Finally, the labor of humans and machines actually execute the operations of the organizational entity. A "topology" of organizational structure has emerged from previous efforts in real-time intelligent systems.

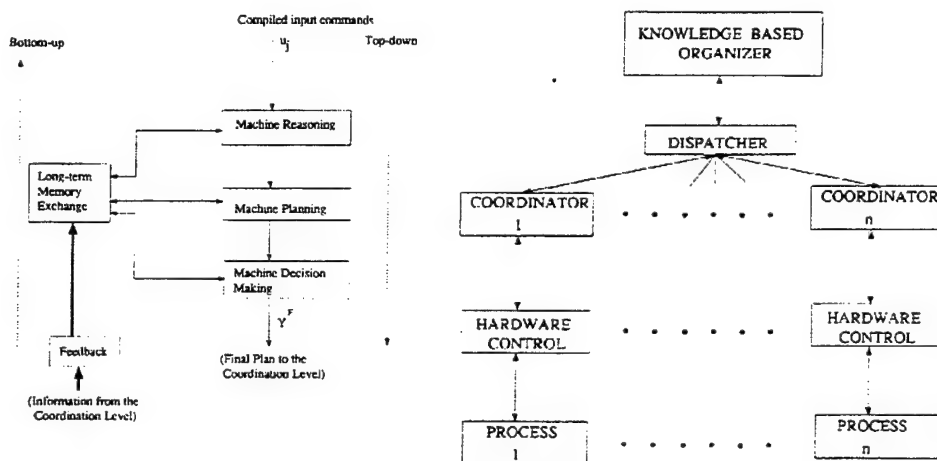


Figure 6

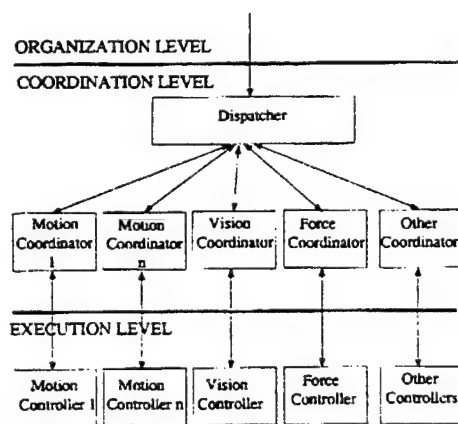


Figure 7

These diagrams illustrate an anthropomorphic model for intelligent real-time systems design. This type of system architecture exhibits a number of properties. First, a hierarchy of processes is defined. The "higher level" processes require more "abstract" processing such as symbol creation and manipulation. "Lower level" processes directly interact with the environment. For Agents, the environment is the message data stream. Secondly, timing requirements tend to be inversely proportional to the level of a process. Low level processes need to be highly responsive. High level processes are often very slow to respond. This layering of timing requirements allows a system to maintain sufficient interaction with its environment while working to improve its capabilities. The "scope" of systems processes tends to be layered in a similar manner. Higher level processes incorporate system-wide and long range data elements, i.e. they process the "big" picture. Low level processes have well defined and narrow tasks tailored to specific increments of time. This leads to a third characteristic of intelligent real-time systems. They are structured such that an "abstract" "high level" goal or set of goals can be designed and understood without explicit attention to the numerous run-time or operational details. The system is designed to map the high level goals into a set of lower levels functions necessary to achieve the given goal(s). Lower level processes, such as control loops, have their own low level goals, such as set points. The specific low level processes performed at specific points in time are determined by, or at least influenced by, higher level processes.

Figure 8 is also from Valavanis and Saridis [60]. This diagram illustrates that intelligent control is the intersection of three separate areas of work. The requirements and objectives of Intelligent Control inherently overlap with Artificial Intelligence, Operations Research, and Control Theory. Each one of these disciplines in turn interact with each other and define a number of specialized subdisciplines. From this organizational perspective, a systems designer can recognize that, if properly managed, a considerable amount of knowledge and talent can be utilized for a real-time system application such as throughput optimization of communications networks. The development and demonstration of this technology insertion capability is central to the Agent concept. A mechanism for efficiently and effectively inserting new capabilities into operational legacy systems is a primary objective and focal concern.

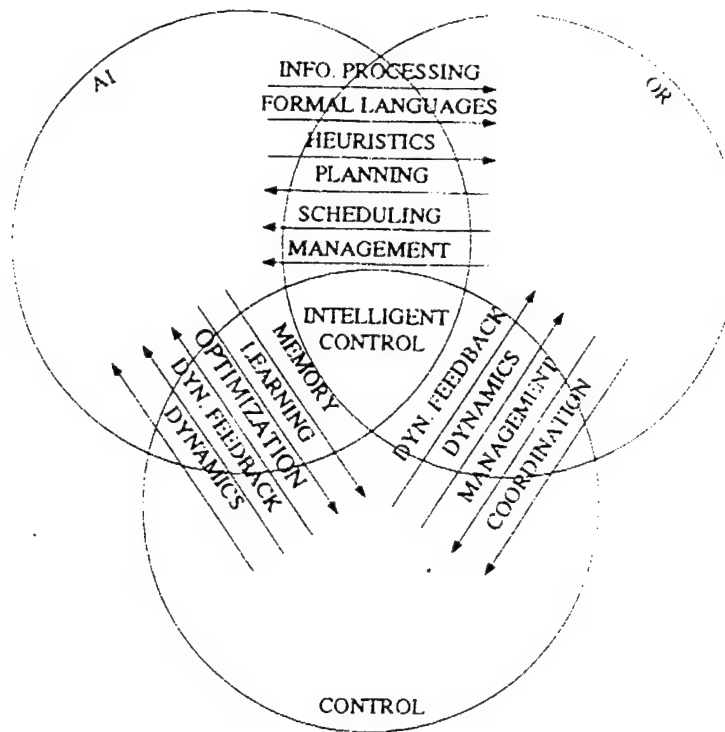


Figure 8

3. Real-Time Expert Systems

Expert systems are not typically real-time systems in terms of their inherent architecture. The heuristic search algorithms typically employed are not guaranteed to meet hard real-time requirements. A number of intelligent control systems employ "Expert System" algorithms and perform within their requirements. Expert systems have become an integral component of intelligent systems, but their application tends to be best for "organizational" processes. Fuzzy logic expert system techniques have been recently demonstrated and they may prove to be quite valuable at the coordination and execution level of the Agent processes [61].

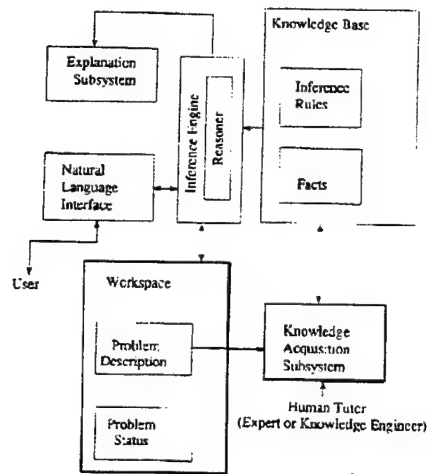


Figure 9

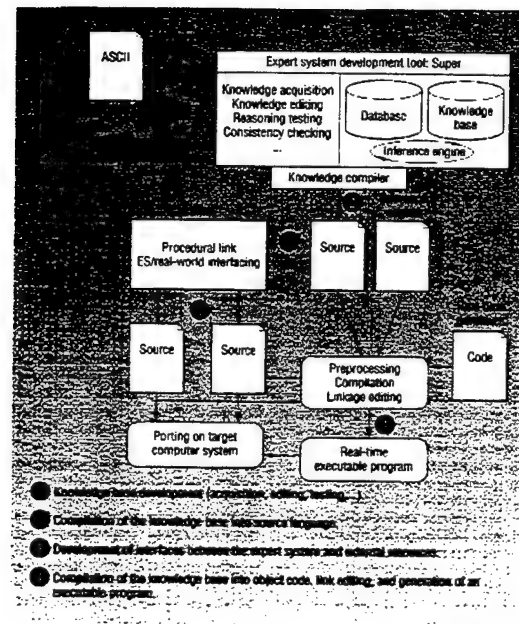


Figure 10

J. OTHER CRITICAL TECHNOLOGIES

1. Information Theory: Mathematics of Communication

a. *Definition of Information*

The following is from Hamming [49];

Suppose that we have the source alphabet of q symbols s_1, s_2, \dots, s_q , each with its probability $p(s_1) = p_1, p(s_2) = p_2, \dots, p(s_q) = p_q$. When we receive one of these symbols, how much information do we get? For example, if $p_1 = 1$ (and, of course, all the other $p_i = 0$), then there is no "surprise," no information, since you know what the message must be. On the other hand, if the probabilities are all very different, then when a symbol with low probability arrives, you feel more surprised, get more information, than when a symbol with a higher probability arrives. Thus information is somewhat inversely related to the probability of occurrence.

From an information theoretic perspective, "surprise" is associated with the probability of a communication event. Data can be transmitted but unless there is a lack of predictability, there is no surprise and therefore no information content. Agent based communication links capitalize on this understanding of information content. There are a number of ways that the degree of "surprise" can be taken out of network messages. Traditional data compression techniques capitalize on a relatively small number of possible approaches. Traditional data compression, and coding theory in general, will be exploited and incorporated within the Agent architecture, but this is not the technology being developed.

Agent based communications should not be confused with data compression technologies. Through efficient management and accounting of what is known, or can be known before messages are transmitted, agent based communication links share the knowledge in a manner such that there is less "surprise" in the messages transmitted. Thus, Agents are able to communicate such messages with fewer bits.

By incorporating engineering and design knowledge of the platform tracks, the behavior of the tracks becomes predictable and, thus, less of a surprise from one message to the next. Agent based communication also provides a mechanism for incorporating command and control

(C2) knowledge. For C2, different track types typically have different operational requirements for position resolution and update rate resolution, the operational requirements for actual message transmission times and resolutions can be known before messages are transmitted, and therefore, the track messages can be communicated at a lower rate with fewer bits per message. Also note that redundancy can be added when operationally required for increasing the confidence that messages will be received.

Agents are network processes that have a primary goal of identifying and sharing system knowledge that is not shared by the current communication links. One of the assumptions, as stated earlier, is that with minimal impact, such a collection of processes can be inserted into an existing communication network. With Agents, otherwise unutilized knowledge is then used to take the "surprise" out of the messages as they were originally communicated. Agents exploit knowledge within the scope and bounds of typical data compression methods, but data compression is not the technology under development. Data compression related knowledge is considered a relatively small portion of the system wide tactical knowledge available for maximizing information throughput. By identifying and accounting for system wide knowledge, more robust and effective communication is realized *while* increasing the effective throughput of the tactical network. Agents, by design, are background network processes which continuously work to identify and manage tactical network knowledge for achieving their own goal of maximizing effective throughput.

b. Measures of Information, e.g. Entropy

Claude Shannon's paper in the 40's established a mathematical basis for the theory of communication [56]. Shannon defined a mathematical function, called entropy, for measuring information content. Shannon's entropy function is defined as

$$N \sum_{i=1}^q p_i \log \left(\frac{1}{p_i} \right)$$

where N is the number of symbols communicated, q is the number of possible symbols, and p_i is the probability that the i -th symbol is in the N length message. As noted in Hamming [49], "It is important to realize that a remark like 'Consider the entropy of the source' can have no meaning unless a model of the source is included. Your estimate of the entropy of a source of symbols therefore depends on the model you adopt of the structure of the symbols."

A variety of metrics exist for defining and determining information content. Kapur [51, 52] provides a number of examples and references. There is no single absolute measure of information. Entropy and working model of the source is necessary. A model is needed to specify the symbol set and probabilities for the given symbols for all points in time. Usually, the probability distributions are assumed to be "stationary." This means that the statistics of the symbols do not change with time. Operationally, nonstationary data is quite common. Thus, potential performance of "optimal" data compression schemes such as Huffman coding, cannot be realized. Alternative schemes, such as Lempel-Ziv dynamic dictionary based approaches, often demonstrate better operational performance. Thus, there is no one ideal technique or underlying model for a given coding scheme.

Ideally, a more abstract "context dependent" network process would dynamically identify and select the appropriate measures of information and their most appropriate models. Such metrics and models can then be selected based on the degree of compression which can be operationally realized at a given point in time. While managing and exploiting other types of knowledge, such as "system design knowledge" and "C2 knowledge," Agent processes can also manage "data compression knowledge." Thus, improved operational performance of traditional data compression schemes is inherent in the Agent architecture.

c. Universal Compression and Retrieval

The following summary note is from Krichevsky [50]:

The output of a source may be compressed up to its entropy, not more. It is the main fact of the source coding theory, which has its origin in Shannon (1948). The first theoretical compressing code is Shannon's. It happened to be very close to Morse's, which was developed empirically a century earlier. Shannon's code is very simple, although not optimal. An optimal code for a stochastic source was constructed by Huffman (1952). It may be used to compress an individual word. The frequencies play the role of probabilities. The encoding may be either two-pass or one-pass. In the first case the

frequencies of letters are counted beforehand, then a code is built. In the second case the code is changed along with the word reading.

Output words of Shannon or Huffman's codes are of different lengths. There is a compressing code, whose output words are of equal length. It was developed by first G. L. Khodak (1969) and then by F. Jelinek and F. Schneider (1972).

A decipherable code for integers is constructed by V. Levenstein (1968) and P. Elias (1975).

Hansel (1962) and Krichevski (1963) exploited the source coding to lower-bound the length of threshold formulas.

The Kolmogorov complexity bridges the theory of algorithms and the information theory. Any code, including Lempel-Ziv's, move to front, etc., is only a majorant of Kolmogorov's.

In other words, majorants of Kolmogorov codes provide a theoretical framework for compression algorithms. In this context, optimality is tied to computability, as well as, compressibility. Computing cost is typically a critical operational parameter, but until recently, it has not been an integral theoretical component.

Krichevsky also notes that "for many years the target of the source coding theory was the estimation of the maximal degree of the data compression. This target is practically hit today. The sought degree is now known for most of the sources. We believe that the next target must be the estimation of the price of approaching that degree. So, we are concerned with the trade-off between complexity and quality of coding." The reader may need to be reminded that in this theoretical context, the sources are assumed to be known and well behaved.

Computing time is a critical factor for agent based communication links but until recently it has not been a theoretical focus of concern. Accurate and robust source models are another critical operational concern that is often overlooked by information theorists. Krichevsky's analysis assumes that the sources are known and well behaved. Temporal variability is not within the context of his work.

Within its own scope and context, coding theory provides a basis for determining compressability. With full knowledge of their utility and limits, compression algorithms can be identified and utilized at the most appropriate times. As the characteristics of data sources change with different tactical situations, software agents can activate the most appropriate

compression algorithms and source models for the particular context. Agent's can operate on knowledge within and beyond the scope of coding theory and data compression. How these bodies of knowledge are coordinated is of particular concern for optimizing the operational throughput of a network.

d. Universal Prediction, e.g. Machine Learning, Generalized Approximation, etc.

Universal prediction is another recent focus in coding theory. If a source is well behaved and can be modeled, then the source should become known over time and the sequence of symbols should be optimally predictable. Theoretically, if the symbols can be predicted, optimal coding schemes should incorporate such predictability. Traditionally, predictability is not assumed. For adaptive schemes, lack of predictability is the alternative focus.

Universal predictive coding schemes are rooted in the same mathematics as related areas, such as machine learning, generalized approximation, pattern recognition, classification, and adaptive filter theory. Probability and mathematical statistics are typically developed and applied using the numerical methods of linear algebra. When possible, these techniques often combine empirical and analytical modeling methodologies. Analytical models are preferred over empirical "black box" approaches.

Just as the Krichevsky and others are interested in bridging complexity theory and coding theory, those working in the area of universal prediction and interested in bridging the mathematics of prediction and coding theory. The Agent approach under development is not limited to coding theory, or those areas of coding theory most related to efficient network communications. As previously noted, coding theory has its limitations and Agents address those limitations while exploiting opportunities which are not within the scope of traditional coding theory approaches.

Coding, complexity, and prediction can each be exploited by a more abstract system process. At this level of abstraction, a spectrum of competing data compression algorithms can be evaluated. This can be done alongside competing prediction algorithms. For real time execution, algorithmic complexity is a critical factor for both coding and prediction. The Agent architecture is designed to manage these traditional components, but these plug-in capabilities could probably be managed by a number of other more traditional techniques. The ability to

identify and quickly utilize system wide knowledge, is the focus of agent based communication links, i.e. Agents. The agent based architecture provides the mechanism for incorporating additional bodies of knowledge for maximizing network throughput.

Figure 11 illustrates the relationship between pattern recognition and data compression. In both cases, invariances are of special interest. The goal is to identify functions whereby predictable and recognizable patterns can be identified and exploited. A number of "plug-in" pattern recognition capabilities have recently become available. Agents are being designed such that those capabilities can be managed and exploited. The design objective is to demonstrate "coordination" and "organizational" network processes that manage C2, system design, data compression, pattern recognition, and other bodies of knowledge for the purpose of maximizing the effective throughput of tactical communication networks currently operating in the Fleet.

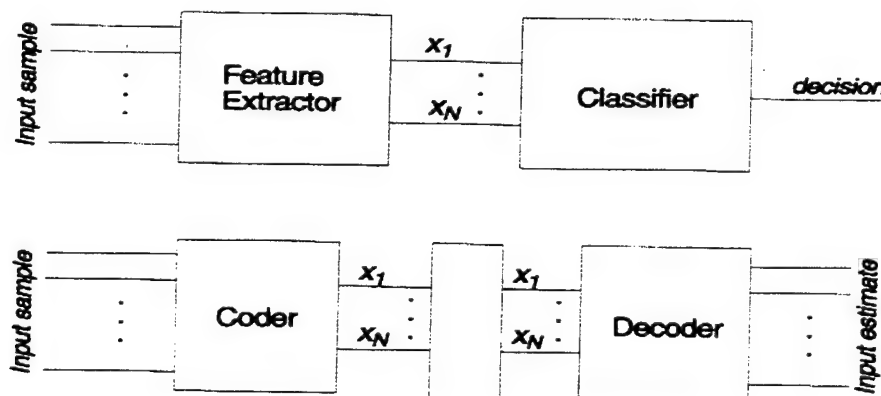


Figure 11

As mentioned earlier, universal predictive coding is inherently related to machine learning. Intelligent real-time systems share a similar interest in machine learning. The objective for hierarchical intelligent systems is to design systems such that they can learn to create optimal tasks and coordinate their execution. The following is from Lima and Saridis [62] and discusses work related to this objective:

A robot can learn from the data provided by its external sensors (such as cameras, ultra-sound transducers, or proximity detectors) or internal alarms (such as a battery failure or a time-out while a process runs). Often the goal is to learn how to recognize a scene or an object in the scene, by either a statistical or a structural approach. The robot explores the similarities of the scene or object with previously stored patterns. Some learning programs record the strategy used to solve a particular problem so that they don't have to

search for a solution when the problem emerges again [Samuel 1963]. Other programs, such as those based on neural networks, begin with a random network and continually change the connections (or weights) among network nodes, to reflect a bad or good performance as they try to accomplish their job [Hertz 1991]. This later type of learning algorithm is usually classified as unsupervised if it receives no information about the correctness of its output, and as supervised if the goal is available. We can improve the algorithm's performance by measuring the error between its goal and its actual output. A third category is reinforcement learning, where limited information is available about the algorithm's instantaneous performance, typically in the form of success or failure signals.

Since the late sixties, various strategies based on reinforcement learning have emerged to address the control of complex systems. Reinforcement learning is particularly interesting for robotics, where it involves the exchange of small bandwidth information (failure or success) between robotic subsystems. In typical applications, such as unmanned space or underwater missions, the cost of large bandwidth for communications between the central command (earth controller or main-vessel controller) and the vehicle is prohibitive. Designers can reduce this cost by increasing the autonomy of the machine involved in the mission.

Fu was perhaps the first to write about learning control systems and to define intelligent control systems as systems of an interdisciplinary nature where artificial intelligence and automatic control intersect [Fu 1986]. Moreover, he introduced the concepts of *stochastic automata* and *stochastic grammars*. Narendra and his associates have also developed work on stochastic automata [Narendra and Thathachar 1989]. In the last few years, Sutton and his associates have explored reinforcement learning solutions that associate these two views of stochastic automata [Sutton 1988].

A frequent limitation of reinforcement learning applications to robot problem's is the problem's large state space. Lin attempted to tackle this problem by providing initial knowledge to the robot and by endowing it with generalization capabilities via neural nets [Lin 1994]. Sutton described an algorithm that learns from both virtual experiences in an internal-world model and real-world experiences, to accelerate the learning process [Sutton 1990]. Both of these authors used Watkins's *Q-learning* algorithm, which includes a learning-performance function [Watkins and Dayan 1992].

Finally, Wang and Saridis used reinforcement learning to improve performance at the intermediate level of the hierarchy [Wang and Saridis 1993]. And McInroy and Saridis proposed reliability as a practical measure of primitive-task performance – but did not consider the computational cost [McInroy and Saridis 1994].

The above discussion illustrates that neural nets are but one technique of machine learning. In practice, neural nets are more accurately described as a family of regression techniques utilized to generate empirical models. When there is limited communication bandwidth between agent processes, e.g. mother ship and vehicle, agents can minimize communication requirements. For purposes of the Agent architecture, agents with synchronized

models only need to communicate when *one* of the agents recognize a deviation from the “real world,” i.e. part of its environment not shared with the other agent. Furthermore, since the goal is to minimize communication requirements between such agents, synchronized learning algorithms are another body of knowledge to be exploited.

III. ENGINEERING DESIGN FOR A NODE AGENT

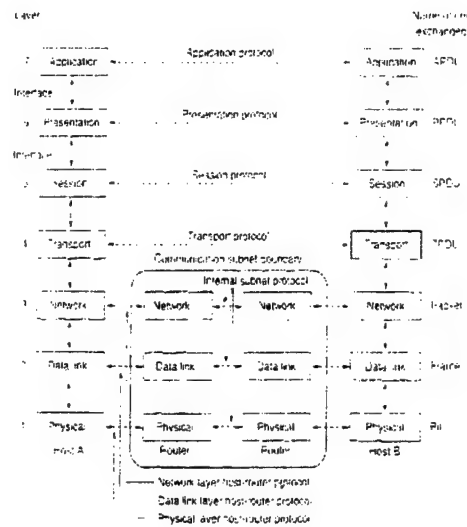


Figure 12

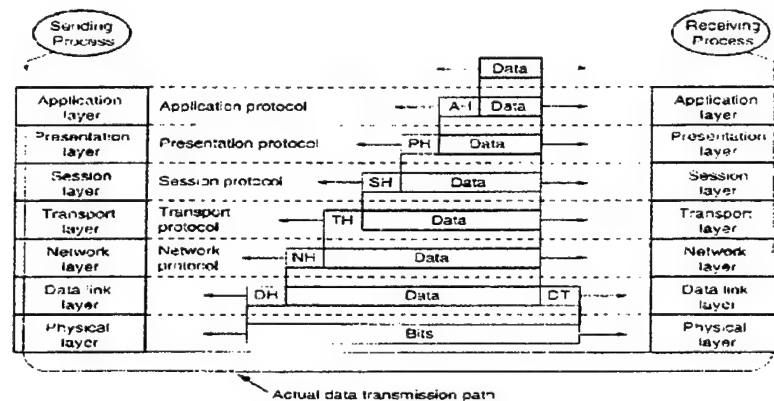


Figure 13

A. CODESIGN: SOFTWARE ENGINEERING FOR REAL-TIME SYSTEMS

Delays in communication are a primary concern for tactical networks; just as transcontinental phone conversations were plagued with annoying delays due to the limitations of hardwired signal transmission. The addition of any process in the path of C2 messages is an even greater concern. Because Agents map messages and communications to more efficient representations and transmission timing, they inherently introduce delays on a per transmission

basis. If Agent processes are introduced at the 1553B bus, very small latencies may be required. This may be a significant constraint. If the Agent processes are designed into the CDLMS, much flexibility will be gained, but message management will still delay message communications. For any communications network, messages are typically delayed due to typical management functions. The critical concern is that such delays be minimal. For time critical data compression applications, data compression processors (DCP) are typically incorporated into the system hardware. This is a "plug-in capability" that our agent-based approach can manage in the course of minimizing message traffic while maximizing effective throughput. Other algorithm specific application processors (ASIP's) are commercially available. Agent's provide an opportunity to utilize those off-the-shelf plug-in capabilities, as well as.

Several domains such as embedded, real-time, and reactive systems are the application areas for which hardware / software codesign techniques are most beneficial. They are beneficial because the increasing complexity of advanced systems combined with technological advancement requires new design methods and integrated tool environments.

Hardware / Software codesign is different from conventional approaches in that it continuously relates the hardware development cycle to the software development cycle. Hence, hardware decisions significantly affect software design activities and vice versa. In codesign, the entire problem is treated as a whole. The "co" means primarily *together*. However, it also expresses a design flow that is properly coordinated; a joint effort among designers from different areas. The effort is inherently concurrent in that the majority of all design steps are carried out in parallel by a team that guides, coaches, and supervises the design process. Ideally, several teams of experts develop system components rapidly and resolve problems in a timely manner.

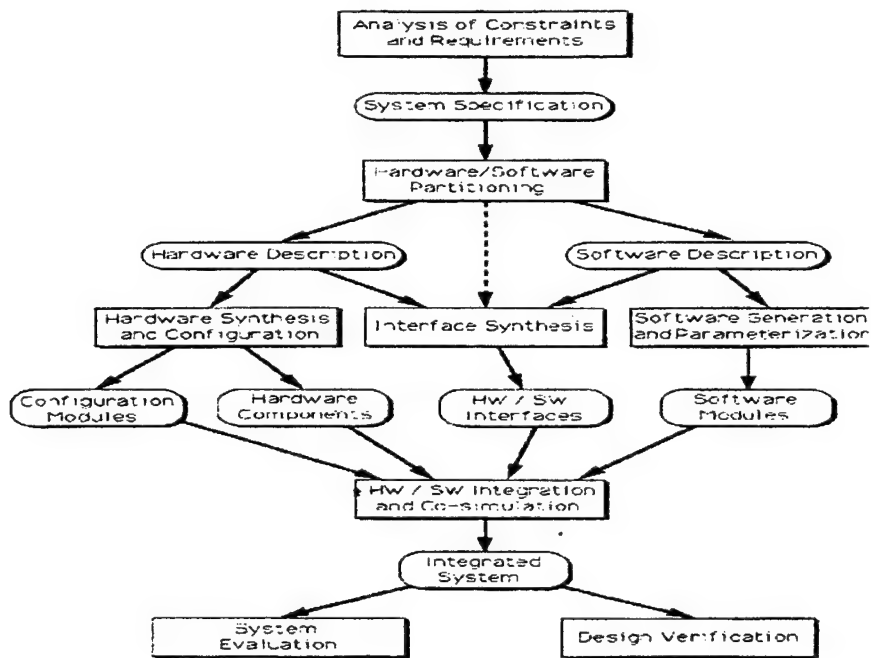


Figure 14

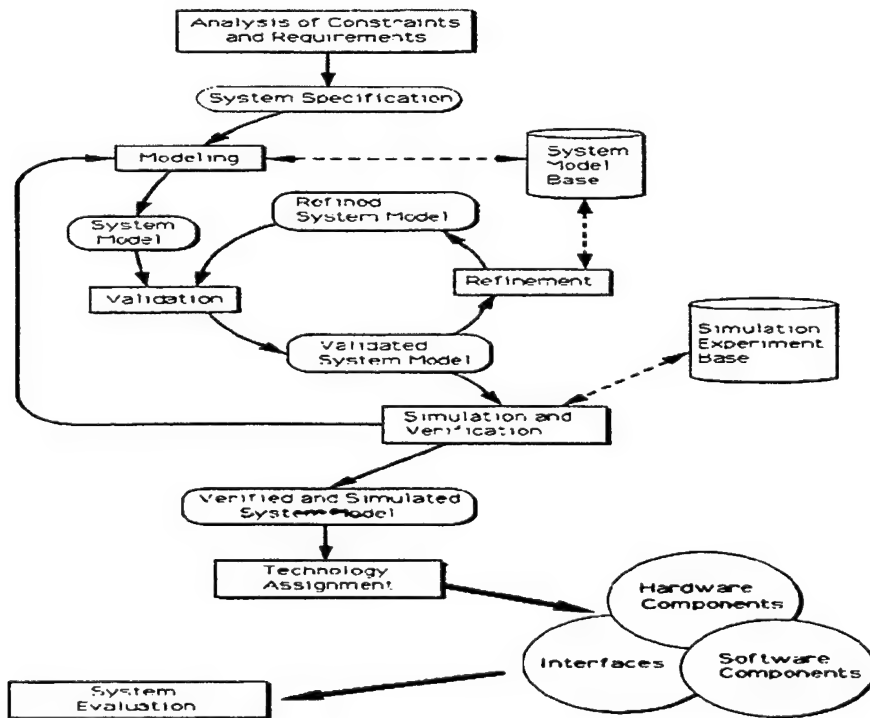


Figure 15

B. AN AGENT FOR LINK 16/22

1. A Concept

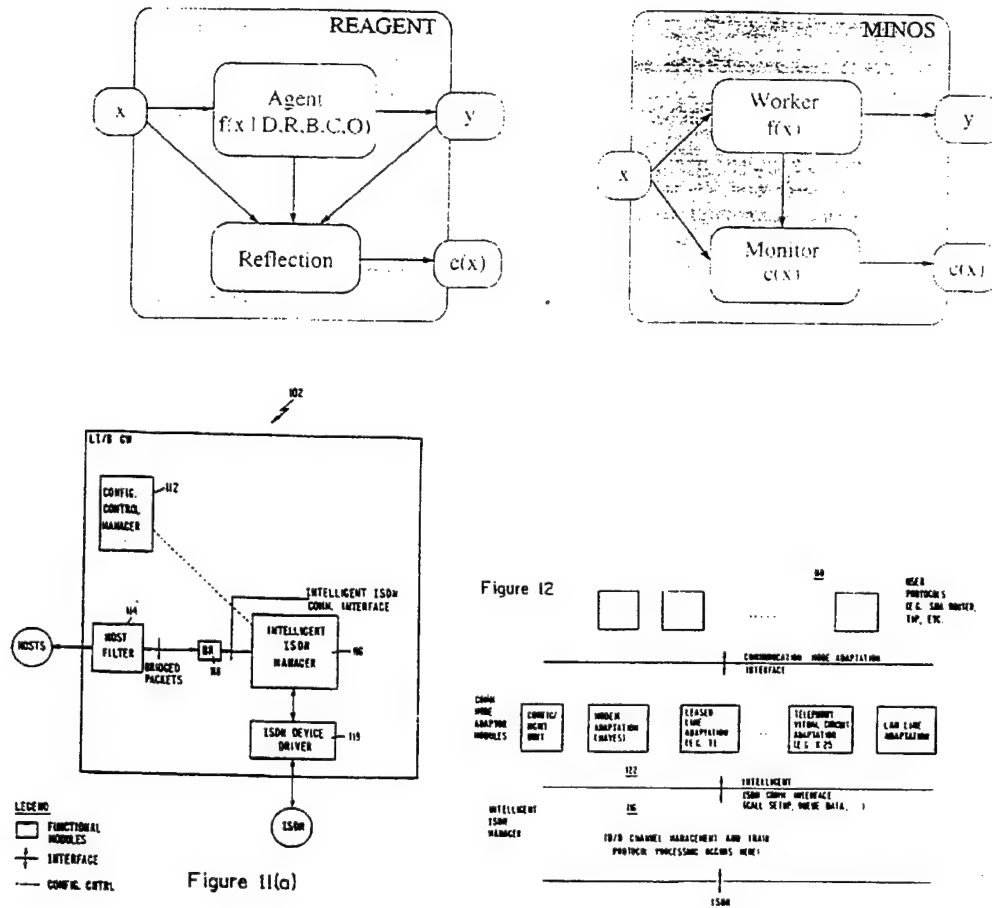


Figure 16

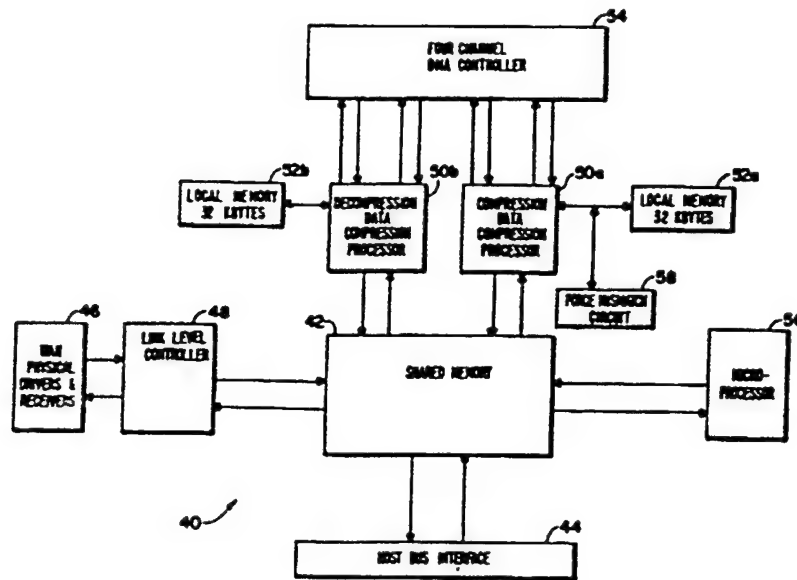


Figure 17

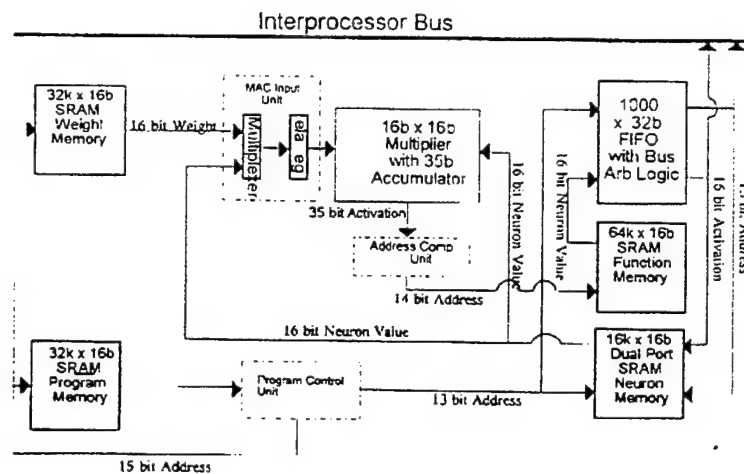


Figure 18

C. SOFTWARE REQUIREMENTS - A FUNCTIONAL DISECTION LEADING TO A PROPOSED DESIGN FOR IMPLEMENTATION

- Agent will not be specific to any particular digital network, but will be appropriate for any digital network where it can intercept and alter messages to and from all nodes on the network.

- Agent will be a software agent in structure.
- Agent will be able to run under any common operating system.
- Agent, in the engineering design, does not require any particular computer language or programming style.
- Design & Architecture must support modular concepts of engineering and be adaptable to future enhancement / expansion.
- The design of Agent should be as general as possible within the constraint of performing within the tactical network environment and Link-16.
- The software and design should be readily adaptable to major link component upgrades, as well as future systems and technologies.
- Agent control should be distributed rather than centralized.
- The Agent must be instantiated on every node in the link.
- The Agent must not require a special or central node for control or any other purpose.
- The Agent can continue to operate correctly with the loss of a number of transmitters or receivers.

1. Functions & Operation

- Agent will be able to perform all known heuristics (e.g., delta messaging, update bundling, extrapolated updates, and so forth), and be extensible to other analogous techniques as they are developed.
- Agent must decouple its processing components from its knowledge of link rules and operations, so that a simple substitution of link rules and operations can enable it to work with other tactical links (e.g., Link-11, Link-4A, Link-22).
- Instantiations of Agent will be able to communicate with each other in order to optimize their operations.
- Communications between instantiations of Agent should not be visible to host processing systems and the C2P (JTIDS, of course, will be aware of these transmissions).
- Inter-agent communication should not in itself constitute a significant burden to link operations (that is, should consume relatively little bandwidth or link

overhead).

D. FUNCTIONAL REQUIREMENTS

These requirements are specific to Link-16, but many may apply to other digital networks.

1. Agent will increase the effective bandwidth of one or more Link-16 NPGs.
 - a) Agent will be able to measure the message loading within an NPG.
 - b) Agent will be able to detect when the message loading reaches, or is about to reach, saturation.
 - c) Agent will be able to selectively employ an heuristic or combination of heuristics to reducing the loading within an NPG (and, by so doing, increasing the effective bandwidth).
 - d) Agent will rely on Time Slot Reallocation (TSR) to distribute the additional effective bandwidth to other purposes, including RTR messages.
2. No "rules of the link" will be violated by Agent operations.
3. Agent operations will degrade operations within an NPG to the minimum extent necessary to prevent or delay NPG saturation.
 - a) Agent will employ lossless techniques whenever possible.
 - b) Agent will employ lossy techniques only when necessary to avoid saturation.
 - c) Agent will explicitly manage NPG degradation due to its operations.
 - d) Agent will employ lossy techniques only when such losses are operationally acceptable.
4. Agent will accommodate all current and planned users of J-Series information.
5. Agent will not delay the transmission of J-Series messages beyond the point of military usefulness to the receivers within the tactical context.
6. Agent will be able to function transparently on the bus that leads to or comes from the JTIDS terminal (when implemented in a JTIDS environment).
7. Agent shall work when implemented on Link-16 but not be specific to Link-16. It should readily be reconfigurable to operate within other tactical links.

8. Agent will not fail or degrade if one or more nodes unexpectedly drop off the link. There will be no main or central node.
9. Agent will be able to operate successfully in an environment with degraded connectivity (that is, one or more receivers fail to receive a complete message stream, probably due to loss of line of sight).
10. Agent will be able to measure its own effectiveness and dynamically alter its operations in response to those measurements.
11. Under no circumstances will Agent operations cause an erroneous J-Series message to be received at the host component.
12. Agent will provide a complete tactical picture within militarily-acceptable intervals, which may be context sensitive.
 - a) To platforms recently entering the link.
 - b) To passive listeners.

E. KNOWLEDGE-BASED COMPONENTS

1. General

The knowledge-based components for Agent are especially designed for Agent operations. Many changes from the usual knowledge-based paradigms were necessitated by the need to do more than simply recognize situations, but also to perform actions in response to those situations. Therefore, this section is designed to describe the special properties of the knowledge-based components so that, during detailed design, appropriate tools can be properly designed (e.g., translators from KIF to Agent form, translators from Agent form to semantic networks, and the inference engine).

2. Semantic Networks

Facts are truth assertions used either to create or edit a semantic net, or provide the input to inferencing. They are also used by the metarule functions. It is assumed that the reader has a general knowledge of semantic networks, facts, production rules, inference engines, and other related terms used in artificial intelligence.

a. *The Semantic Network*

Underlying the facts and rules of Agent is the concept of the semantic network. This network is created within global memory available to any of the processes (and processors) of a single instance of Agent. The semantic network is created at agent initialization by the use of facts with an expanded grammar. For example,

```
A message is_a null_class  
The value facet of the max_delay slot of the message is 25 ms
```

The first statement creates a object called "message" and assigns it with an "is_a" relationship to the null class or object (which is created automatically in shared memory). The

second statement creates a slot called “max_delay” within the message object. A component of the initialization process creates this semantic network in persistent store (using IBM’s ABE terminology).

In the present design, this network cannot be added to during agent operations. Although “is_a” statements during initialization create classes, “is_a” statements in rule consequents create instantiations (or objects).

The inference engine “looks up” values in the semantic net when antecedents reference slots in the semantic network, and “sets” values in the semantic net when consequents fire.

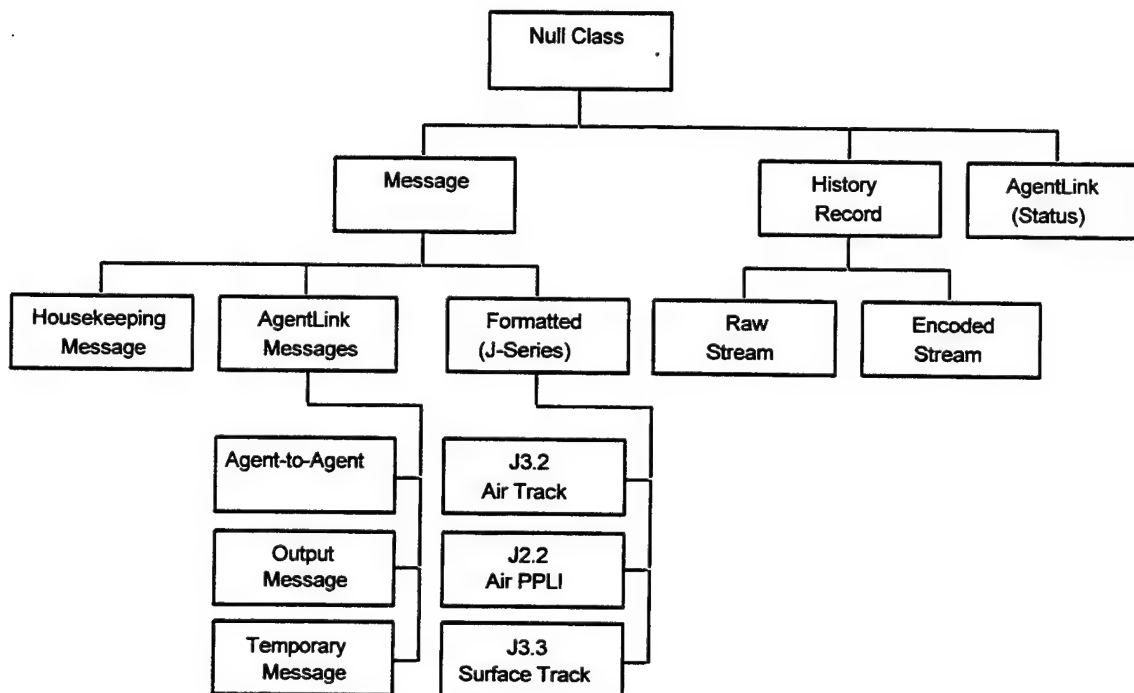


Figure 19, Agent Semantic Network

b. Slots and Objects

(1) NULL slots and the NULL object.

A special object is always present at the top of the semantic network, the null frame (or null class or null object). All classes and instantiations are subclasses of the null frame. It has many uses.

It is awkward to always have to explicitly refer to a slot reference in the rules:

The <slot name> of the <object name> is <value>

To simplify where possible, the <object name> always defaults to the null object. For example,

The rules below use this construction often:

IF Agent is operating

This usage refers to the "Agent" slot of the null object, whose value might be "operating." On the other hand, "The mode of Agent is operating" refers to the mode slot of the Agent object.

(2) Slot References

A slot reference is of the following formats:

<frame>
<slot> [<frame>]
<slot> [of ANY <frame>]
<slot> [in ANY <frame>]
[THAT] <frame>
[THOSE] <frame>

If the name of the slot is omitted, then the default slot of the named frame is referenced. If the name of the frame is omitted, then the null object is assumed. So the following are slot references:

1. The door
2. The door of the house
3. The door of ANY house
4. THAT | THOSE door

In the first case, since the name is omitted, the slot is part of the null object. In the second reference, the slot is part of the house frame. If both are used they reference different values. In the third form, the slot reference is to the set of all slots of all instantiated houses. In the last form, which is always paired with one or more ANY constructions, the reference is to all slots referred to by the matching ANY constructions.

(3) Dynamic object and slot creation

Note that these constructions do not create slots or objects in antecedents, but do create slots and objects in consequents. If there has been no statement such as

THEN Agent is operating

there is no "Agent" slot in the null object when

IF Agent is operating
is encountered by the inference engine. Therefore the test fails and the slot is not created.

(However, the slot could have been predefined and initialized during Agent initialization—the point is that it doesn't have to be.) This kind of dynamic object and slot creation makes it much easier to write rules where perishable knowledge is involved, and, in fact, is the preferred way of writing such a rule.

F. FACTS

1. General Forms

Facts in Agent are references to a semantic network. Facts consists of the following forms:

<slot > [of <frame>] <declarative operation> <value>
<slot > [of <frame>] <relationship> <frame reference>
[<facet >]of <slot > [<frame>] <declarative operation> <value>

The <frame> refers to the class or object itself, the <slot > is the name of a slot within that frame. The <facet> refers to a facet within a slot. Items within square brackets are optional—when omitted, the reference defaults to the null object. Facet references are only used in building the semantic network, which takes place at Agent initialization time.

Sample declarative operation:

is

Sample relationships:

is_a
has_a

Below are a set of sample facts, with their forms in parentheses:

The default_facet of the weight slot of the airplane frame is 32,000 lbs	(3)
The house is green	(1)
A house is_a building	(2)
A building has_a door	(2)

Note that there are a number of words used for readability that are completely ignored by Agent, such as “the” and “a.”

In a later section of this report, the use of facts in rules is explained and differences in syntax and semantics are noted. This section does not explain how the various forms of facts are used. It is often helpful to have plural forms of slot or frame names. When defining a semantic net, for example, the following form can be used:

The house(s) is_a building(s)

This creates a house frame with a synonym for house being houses. There is no meaning associated with plural forms—they are used precisely as singular forms.

2. FactSets

FactSets are collections of facts referenced by name. See under MetaLanguage.

G. PRODUCTION RULES

1. General Forms—Antecedents and Consequents

The general forms of Agent production rules are as follows:

1. <slot reference> <operation> (<value> [<units>]) [ONLY | ALSO]
2. <slot reference> <comparison> <value> [<units>]) [ONLY | ALSO]
3. <slot reference> <comparison> <slot reference> [ONLY | ALSO]
4. <directive> <slot reference> | <frame reference>

2. Comparisons

The following comparisons are defined in Agent:

is - a positive statement of truth
isn't - a negative statement of truth
is_greater_than - >
is_greater_than_or_equal_to - >=
is_less_than - <

is_less_than_or_equal_to - <=
 is_equal_to - numeric equality
 is_not - logical negation
 is_identical_to - non-numeric equivalence
 is_the_same_as - non-numeric equivalence, less forceful than equality
 is_different_from - non-numeric non-equivalence, less forceful than is_not

These all have meanings which are applied on a concrete, and abstract formulation of comparisons.

3. Antecedent format

An antecedent appears in the following format:

1. <slot reference> <comparison> <value> | <slot reference>
2. There is <object>

The following are examples of antecedents of the first form:

IF The color of the house is green
 The color of the house is_the_same_as the color of the barn
 The mode is delta_messaging
 THAT message is_a J3.2

These antecedents, when referenced, find the appropriate slot reference, perform the comparison, and return a truth value. It is important to note that they change nothing. In the last antecedent shown above, the “is_a” checks to see if any message signified by the THAT construction is instantiated as a J3.2 message (or instantiated from one of its subclasses). An object can have an “is_a” relationship to any number of classes.

Each antecedent is implicitly connected by logical conjunction. However, an OR construction can be used when parentheses are added:

IF (The color of the house is green OR
 The color of the house is red)
 The color of the house is_the_same_as the color of the barn
 The mode is delta_messaging

This form, when compiled, results in two separate rules with no OR construction.

Antecedents of the second form are special. The parser, upon seeing “There is” beginning an antecedent, checks to see if there have been any instantiations of the object references, and

returns TRUE if there have been, FALSE otherwise. In addition, there is an implicit ANY involved, in that subsequent uses of THAT will refer to the instantiated objects.

4. Consequents

<slot reference> <operation> <slot reference> | (<value> [<units>])

The operations in consequents change the values of slots or create new relationships. The defined operations are as follows:

is
is_a
has_a

Consequents are executed in the order given.

5. Sample Complete Rule

IF	(The color of the house is green OR The color of the house is red)
	The color of the house is_the_same_as the color of the barn
THEN	The house is desirable The price of the house is unknown ONLY

This rule is provided to illustrate many of the bad things one can write in rules. For example, no specific house is referred to. Therefore they will fail since they do not refer to instantiations. The rule should have been written as follows:

IF	There is a house There is a barn (The color of THAT house is green OR The color of THAT house is red)
	The color of THAT house is_the_same_as the color of the barn
THEN	The house is desirable The price of the house is unknown ONLY

In this version of the rule, the consequents are vague. Since, in the first consequent, we refer not to "THAT house" but, simply, "house," the value "desirable" is concatenated (see below) with other values in the default slot of the house frame, if there is one. Since this is a

modification to declarative knowledge, all houses would default to “desirable” as a result. Similarly in the second consequent. Here’s another try:

IF	There is a house There is a barn (The color of THAT house is green OR The color of THAT house is red) The color of THAT house is <u>the_same</u> as the color of barn
THEN	The appeal of THAT house is desirable The price of THAT house is interesting ONLY

Note that the first two antecedents identify the set of instantiated houses and instantiated barns. But the “color of the barn” reference will probably cause a problem since what we are probably interested in is the set of houses that have barns of the same color. So, finally, here is the form that seems correct:

IF	There is a house THAT house has_a barn (The color of THAT house is green OR The color of THAT house is red) The color of the barn of THAT house is <u>the_same</u> as the color of THAT house ²
THEN	The appeal of THAT house is desirable The price of THAT house is interesting ONLY

Note that “has_a” relationships amount to a special type of slot reference (special in that all of the slots and other linkages of the barn frame are now part of the house frame).

H. CONCATENATION IN THE RULES

Multiple values can be assigned to the same slot simultaneously.

THEN The house is green
does not refer to any house in particular (it isn’t “THAT house is green”) and there is no object reference. Therefore a slot is created in the null object called “house” and its value is assigned to be “green”. (Slots are automatically created when referenced—they do not all have to be predefined in the semantic net creation step of agent initialization. They are never removed.)

² Two uses of THAT in the same antecedent cause a pair-wise comparison.

THEN The house is green
 The house is big

The above concatenates two values in the house slot of the null object, "green" and "big."
Therefore,

IF The house is green
succeeds, because "green" is one of the slot values.

IF X
THEN The track of that message is xxxxx
 The track of that message is yyyyy

In this construction, the message referred to as "that" message now carries both track IDs. This is because the slots of the object³ concatenate values assigned to them rather than replacing them by default. A query as to the track ID will match as long as one of the concatenated values match.

If it is desired that only one value be present in a slot, there is a directive for that in the production grammar: ONLY. It looks like this:

IF X
THEN The track of that message is xxxxx
 The track of that message is yyyyy ONLY

Of course, the assignment of xxxxx wouldn't actually be in the real rule if ONLY was meant. When a consequent ends in ONLY then all concatenations are removed and the single value is assigned to the slot.

The temporary message is where all operating processes/processors assemble a batched message. There needs to be only one of these, as each process/processor will contribute to a single output message. They only need to lock that message while they add to it. Also, if the message is long enough, or the oldest update has aged enough, the message becomes an output message and is deinstantiated as a temporary message. Any process with the authority of adding to the temporary message can make this switch. Also, this is how the exception handler outputs a message when it senses and exception condition.

³ An "object" can either be a class (frame) or an instantiation of a class (frame) in this document.

I. ANY/THAT/THOSE CONSTRUCTIONS

1. General

When ANY appears in an antecedent, the inference engine creates a list of all objects that satisfy the specified conditions (if any). Thus,

IF The status of ANY message is late
will create a list of the internal identifiers of all late messages. The THAT (or THOSE)
construction then refers to all items in that list. Therefore,

IF The status of ANY message is late
THEN The status of THAT message is output

changes the status of every instantiation of message with "late" status to "output" status.

If there are no objects (instantiations), then ANY fails.

2. Subsets

IF The status of ANY message is late
 The type of ANY of THOSE message is video
 The time of ANY of THOSE message is today
RULE THOSE messages are no good

In this instance, the THOSE in the consequent refers only to "late video messages generated today." In this construction, the antecedents have order (each subsequent ANY creates a subset of the previous ANY). However, in the consequent, only the smallest subset is referred to. This is the only instance in the grammar where the antecedents must be ordered. Therefore it may take several rules to inference properly.

3. Evaluation Order

A slot has one or more facets. Every slot has the value facet, which, unless initialized when the semantic net is initialized, is initially empty. However, the convention in Agent is that there are two other facets in every slot: the default slot and the if_needed slot.

The default facet contains a value that is used if the value facet is empty. In general, rule bases do not alter the default facet. There is a value in the value facet and a rule causes that value to become empty, the inference engine will use the default value at its next opportunity, if there is a default value.

Finally, an "if needed" function belongs to the third standard facet. If the first two facets are empty, and there is a function in the if needed facet, that function will return a value. So when referencing a slot, the following order is used:

1. value
2. default
3. if needed function

If the rules writer knows that the if needed function need only be called once, the following construct can be used:

The temporary_value is the cost of THAT message

where the "cost" is calculated with an if needed function. Then "temporary_value" can be used until the cost needs to be recalculated. Using if needed functions is best when the value returned must be measured from a constantly changing world model.

IF The distance_from_landing of the aircraft is_greater_than 20 miles

In this case, each time the distance_from_landing slot is accessed, the if needed function provides the current answer.

J. DIRECTIVES

Directives are special words which perform special actions in consequents of rules. They have a different syntax than ordinary consequents. Examples are DEINstantiate, LOCK, UNLOCK, MOVE, INFER, etc.

```
IF          ANY message is late
THEN       DEINstantiate THAT message FROM <frame>
```

1. DEINstantiate

DEINstantiate <object> [FROM <frame>] performs the opposite action of “is_a” in a rule consequent. Instead of creating an instance or object, it deletes it. I know of no standard inference engine which allows deinstantiation, but we have to have it in order to get rid of messages that no longer have any value to Agent processing.

The part of the consequent that follows the DEINstantiate is parsed according to the normal grammar. So that, in the above case, “THAT message” refers to the subset created by the antecedent use of ANY.

If no FROM clause appears, the object is removed from all of its is_a links. If a FROM clause appears, the object removed only from the listed frame. Thus an object, such as a message, can be instantiated in a temporary manner to a class, instantiated to one or more additional classes, and then removed from the instantiation list of the original “is_a” assignment.

2. LOCK/UNLOCK

LOCK <shared memory> | <class> | <object> | <variable>
prevents all access to a sub-tree within shared memory, or to a named variable. Although LOCK can apply to any sub-tree, it generally applies to the whole of shared memory (which is a sub-tree of the null object) or to the instantiations of one or more messages. When other processes use a construction that accesses or changes a locked sub-tree, they can either hang

until unlock or fail. Hanging until unlock is easier to handle and understand than failure, so the design will cause the other processes to pause until unlock.

One tricky part of LOCK/UNLOCK not shown in the rules below is that after the lock succeeds the rule has to be repeated because the lock of some other process could have succeeded before any given lock takes place. Thus the conditions that prompted the lock could no longer apply. Thus, if I locked an aged entry message, I could very well find out that some other process had marked it for encoding (so it was no longer an entry message). In practice that means that those messages identified by the ANY in an antecedent are no longer part of the THAT/THOSE of the consequents after locking. Which is why, after locking, the rule has to be repeated. These repeats are not shown in the rule sets developed for the engineering design for the sake of simplicity, but will be part of the detailed design.

3. **NUMBEROF**

NUMBEROF <frame> returns the number of instantiations (not classes) associated with that frame or any of its descendents. Thus

 NUMBEROF messages
 returns all of the messages that have been instantiated in the semantic network below the "message" frame.

 NUMBEROF THOSE messages
 will return the number of message as subsetted by previous ANY clauses.

4. **NAMEOF**

NAMEOF is used to return the name(s) of instantiated object(s). There are times when it is important to refer to specific instantiations whose names may have been assigned and not stored.

 NAMEOF THAT message
 returns the true message name. NAMEOF is generally useful in conjunction with variables, as in

 name is NAMEOF THAT message

The status of \$name is empty

If NAMEOF matches multiple names, then all are returned.

5. INFER

A rule base can initiate inferencing on another rule base, pausing until the inference engine concludes. This not only makes the rules easier to read, but also eliminates inferencing when it isn't needed. When INFER is completed, the facts instantiated by the call become part of the facts available to subsequent rules in the calling knowledge base. It looks like this:

IF	X
THEN	INFER knowledge-base

It is assumed that the facts needed for the named knowledge base are the ones available to the calling data base (although the named knowledge base can trigger a function that generates additional facts, if it needs to). The resulting facts simply become available to subsequent rules and are returned to the calling base (or to the metarule level) as the results of the inferencing process.

6. PARALLEL

The PARALLEL directive allows certain rules to fire concurrently with other rules if the system has multiple processors available to it. The use of PARALLEL is discussed in the section below concerning automatic rule sorting.

K. SPECIAL TERMS

There are other terms, such as NULL, MINUS, TIME_f and FIELD that are like directives except that they are part of the normal rule syntax.

1. NULL

NULL is used in facts and rules to erase any values assigned to a slot.

The color of the house is NULL ONLY resets the value to the empty set. This usage is completely different and distinct from the null class to which all objects and instantiations are related. In general, if NULL follows “is_a,” it refers to the null frame. In all other forms it refers to an empty value.

Implementation is automatic in that if no value is assigned to a slot called “null” in the null frame, the result will be to return a null value. Expressions that would set a value into a slot called “null” are disallowed.

2. MINUS/PLUS/DIVIDED_BY (Slot Arithmetic)

For example, in the construction,

IF The status of ANY message is entry⁴
The TIME_f minus the time of THAT message is greater_than the
message processing time limit

a slot reference appears on either side and a numeric value of the slot is expected. If the values in the slots are not numeric the antecedent simply fails. If they are numeric, then subtraction takes place and a value results . This value can be stored explicitly, as follows:

The TIME_f minus the time of THAT message is the delay time

This expression puts the difference into the delay_time slot.

The idea works for other embedded arithmetic operations on slots. There is however a different form of arithmetic in the rules, which is described below.

3. FIELD

FIELD (source or destination string, starting position, number of characters)

⁴ These rules cover the problem of an entry message (left or right) that has sat too long without being processed (encoded or decoded).

FIELD represents a substring delimited by a starting position and a number of characters. The source can be an input buffer or a slot value. The starting position starts at 1. If the number of characters field is set to zero, then the all available characters from the starting position to the end are matched and the number of such characters is returned in the third parameter. If the number of characters is set to a positive value, that number of characters is matched. Thus

NCHARS = 0

string of message is FIELD (left_input_buffer,1,nchars)

transfers the entire contents of the input buffer to the string slot of the message frame, and returns the number of such characters to the nchars parameter.

In Agent initialization, the following statements might appear:

A message_format is_a NULL_CLASS .

A J3.2 is_a message_format

The track-position of a J3.2 = is 14

The track-length of a J3.2 is 22 characters

This declarative knowledge, stored in KIF format, is persistent knowledge. That is, it is not normally changed by consequents of rules as they are fired by the inference engine. They are usually part of the invariant part of the semantic net in shared memory. (The inference engine has the capability of dynamically changing persistent knowledge, which means that it is capable of learning, although Agent does not employ this feature.)

In this way, with the use of field, any message can be decoded from its character string format (raw format) to a set of slot values within an instantiated object. The knowledge of how to do this is external to the agent, so that the agent can operate on any set of messages without changing its internals. The only assumption is that the message is in string format and the field positions are fixed. However, a version of Agent can be built that can deal with variable format messages as well as long as sufficient information is present in the message to be able to determine field starting positions (and, possibly, lengths).

4. NCHARS

NCHARS, which takes no arguments, returns the number of characters in the left or right input buffer.

L. VALUE ARITHMETIC

Simple arithmetic is allowed when setting slot values. For example, sometimes a counter needs to be incremented. For example,

THEN The time of the message is the time of the message + 1

takes the time of the message, adds one to it, and stores it back in the time slot. However, in this case the default is ONLY. There is another directive, called ALSO, which performs slot concatenation. This can be rather interesting if misused. If I write

THEN The time of the message is the time of the message + 1
 The time of the message is 23 hrs⁵

Then I will have a single value for the time of the message, 23 hrs, because the parser interpreted these consequents as follows:

THEN The time of the message is the time of the message + 1 ONLY
 The time of the message is 23 hrs⁶ ONLY

The second use replaced the first. If I had written

THEN The time of the message is 22 hrs ONLY
 The time of the message is 23 hrs + 1 ALSO

then

IF The time of the message is 23 hrs
 The time of the message is 23 hrs + 1

would succeed! Recall that in non-numeric value assignments the grammar defaults to ALSO.

A special but important variation of value arithmetic takes the following forms:

<slot> = <slot> <arithmetic operator> <value> | <slot>
<slot> = <arithmetic expression>

The following are examples of this form of arithmetic:

C = C + 1

⁵ The units after a value are stored but ignored. If the right-hand side of a consequent is can be interpreted as a numeric operation, it will be. Any characters after the numeric are considered to be something like units, stored but unevaluated.

⁶ The units after a value are stored but ignored. If the right-hand side of a consequent is can be interpreted as a numeric operation, it will be. Any characters after the numeric are considered to be something like units, stored but unevaluated.

$$C = 23.5/1$$

Thus counters can be employed as needed. Note that the slot reference is in the null object or frame, and that ONLY is implied. As written, these counters are global.

M. CONCATENATION IN NAMES

The use of square brackets denotes concatenation. In Agent, concatenation is used to produce an unique identifier for an object. Thus,

```
n = n + 1
message[n] is_a update
```

increments the counter, concatenates the value of that counter to the string "message," and instantiates the object under the concatenated name. This is more of a bookkeeping facility than a necessary one, in that one could design the semantic network to have multiple objects of identical names, but it results in a difficult design with no advantage. Once the message has been instantiated as message[n], other rule sets will pick it up or ignore it on the basis of the values in its slots rather than by its name.

N. VARIABLES, POINTERS, AND INDIRECTION

Although the use of slots in the null frame is rather helpful, coordinating asynchronous processes requires some kind of local variables in the rules. For example, in the example in the section above, n is a slot of the null frame. However, if two identical processes are operating, they will not be able to use this mechanism easily. Therefore any identifier that begins with an ampersand becomes a local variable available only to the rule base in which it appears.

```
&n = &n + 1
message[&n] is_a update
```

In this case, local space, space not in global shared memory, is allocated for n, and that allocation is only available within the scope of the process that defines it. Within the operations of that process, n is global. That is, it retains its value once assigned. In some sense it is a permanent variable outside of the scope of the semantic network.

Any structure can be stored in such a variable, including messages. The inference engine evaluates the variable to a structure or a value and substitutes that structure or value in the rules. If, however, I assign a complex structure, such as a message, to a variable, a copy of the original is created. The original itself is untouched.

The pound sign is used for pointer variables. One can write, for example,

#n is message[n]

In this case the pointer refers to the original message and not to a copy of it.

The dollar sign (“\$”) is used to indicate indirection. For example, if I use a slot in the null frame to store the name of an object I have no way of getting to the contents of that object.

THEN n is message[m]
 The status of n is idle

The first use of n is as the “n” slot of the null frame. The problem here is that in the second line “n” is meant to be the value of the n slot of the null frame (which is a message). A slot cannot have a slot. Therefore, to indicate that I want n to be evaluated before the rest of the expression, I use a dollar sign:

THEN n is message[m]
 The status of \$n is idle

The inference engine effectively substitutes “message[m]” for “n” in this case, since “message[m]” is the value of n.

O. REASONING WITH UNCERTAINTY

Although there are many forms of reasoning with uncertainty that can be explicitly added to the consequents of production rules, none are required in this application.

P. THE AGENT ENGINE

The inference engine must be custom built for this project. In addition to the constructions documented in other parts of this document, it will have other components

important to real-time efficiency and embedded applications (in this case, operating to support an intelligent agent).

1. Rule Order and the Academic Paradigm

The tradition has been that rules can be added to knowledge bases as they are constructed without regard to the concepts we normally think about in “structured programming.” In fact, relatively arbitrary rule ordering and entry (via rule editor) is often seen as a virtue. To make this feature possible, inferences engines repeatedly cycle through the rules until one complete pass is made in which no new inferences are made. Given a pathological rule ordering, this strategy can make inferencing very slow, which cannot be tolerated in a real-time application.

Nor is it necessary. The rules in the Agent Inference Engine are ordered (normalized) so that a slot in an antecedent never appears before its uses as a consequent. Thus the rule order is always

IF	A
	B
THEN	C
IF	C
THEN	D

and never the reverse:

IF	C
THEN	D
IF	A
	B
THEN	C

In the first case the inference engine examines the two rules and quits. In the second case it has to process the rules three times: the first time fires the second rule, the second time fires the first rule, and the third time fires no rule (and, so, terminates inferencing). A complex knowledge base can therefore be either expensive or cheap depending upon the ordering of the rules. Other than tradition, there is no argument in favor of arbitrary rule ordering. It is simply a bad idea.

Secondly, ordered rules are much easier to read. There is no reason why rules cannot be read from top to bottom in a single pass—which is the well-proven idea behind structured programming. Easy to read, easy to maintain, less apt to have bugs. Importantly, it is much easier to tell when a rule set is complete when it is normalized.

2. Automatic Rule Sorting and Parallel Processing

Note that recursion can span over many rules making it difficult to spot. For this reason, in the final implementation, the rule editor associated with Agent will automatically sort the rules, in normalized order, and detect any circularities in the process. This is not a trivial thing in that rules can recursively activate the inference engine on other rule bases that can instantiate facts that create recursions (or affect the rule ordering).

A certain subset of the rules being sorted must occur in normalized order. On the other hand, there can be rules between elements of the subset in which the order is irrelevant. These rules may be tested for firing concurrently if multiple processors are available. The rule sorting mechanism can insert the directive `PARALLEL` before a group of such rules, and `END PARALLEL` after them. The inference engine can then send each of the rules between the two directives to special instantiations of itself operating on different processors. The `END PARALLEL` directive pauses if not all of the rules in a parallel section have been examined. The result is a much faster engine, better suited to real-time applications. In general a long rule base will have many such opportunities for parallel processing. The sorter makes them automatic.

3. Functions Within Consequents

Another great benefit of normalizing rule bases is that effector functions can be used in the consequents of ordinary rule bases, knowing that they will take action predictably (that is, a function in a rule will be invoked before a function in a subsequent rule). Thus the Agent inference engine can not only recognize situations, but it can also safely take action to alter things outside of itself.

4. Nested Inferencing

As mentioned in the description of the INFER directive, knowledge bases can call other knowledge bases. This nesting can be taken to any depth, as needed. The idea is to partition knowledge bases for clarity and efficiency. For example, a call to INFER presumes various facts, so that the rules being activated do not have to test for their truth. They simply assume that if they are being activated, certain conditions hold. This makes for much simpler rules. For example, within the confines of a Link 16 network, bandwidth availability, as dictated by the spread spectrum architecture of a given network configuration, places limitations upon the volume of data, and periods of transmission and receipt. To that end, the INFER in this instance is that the Agent is to process all information, and forward, without regard to the Link 16 constraints. Another portion of the Link, is available to carry out the required network transmission / receipt functions.

5. Compiling

Rule sets can be compiled in the Agent system or they can be run interpretively. The reason rules can be compiled is that slot references for all dynamic objects are fixed (or can be, if modifications to the class structure of the semantic network during run time are disallowed). Without compilation, in order to find a message instance it is necessary to search through the net starting with the null object. Once compiled, however, the inference engine can go directly to the instantiations. Any null object slots that are predefined can also be referenced in this way. Slot items that are defined during run time require dynamic look up.

The advantage of interpreting the rules is that a standard validation mechanisms can be used, such as automatic rule tracing and query (e.g., "Why did Agent force transmission of that message when it did?"). The compiled version has no such features and is designed solely for efficiency. Thus the interpreted version is used for building and debugging, while the compiled version is used in the fielded versions of Agent.

6. The WHILE Construction

There are times when it becomes necessary to have the inference engine iterate over a set of rules in a knowledge base. The WHILE construction allows this. Its form is as follows:

```
WHILE    <truth expression>
BEGIN    [name]
          <rules>
END      [name]
```

While the expression remains true (e.g., “The house is red”, “The value is greater_than_10”), the rules between BEGIN and END are iterated. An EXIT appearing within the scope of BEGIN and END causes the WHILE to be retested—it does not exit to the meta controller.

The WHILE construction can be nested indefinitely for complex processing. In Agent, where many updates can be bundled within a single Agent message, the WHILE construction enables a single set of rules to decode all of the updates without having to return to the meta language level.

The name on BEGIN or END is entirely optional. Its use enhances readability. However, if a name is given on the BEGIN the identical name must be given on the corresponding END.

7. BREAK

BREAK [string] causes a jump to the test of a WHILE statement, or an exit from a knowledge base to the meta level if not within a WHILE statement. The string is optional (for readability).

Q. SHARED MEMORY

Shared memory references that common memory utilized by an Agent on a singular node. The processes of an Agent communicate through global, shared memory (RAM). Both persistent and perishable knowledge is stored in shared memory. Persistent knowledge is created at initialization with a special form of facts, while perishable knowledge is created by the firing of rules during inferencing.

There are many uses of shared memory. For example, shared components of the architecture maintains the operating mode, network loading factors, and encoding scheme in use in shared memory. The Shared Component is the only process that sets these slot values within shared memory. All other components simply reference them.

Also, if there are multiple processors operating, any available processor in the delta components can take responsibility for processing a given entry message simply by locking it and changing its status.

Since output messages to the terminal must be batched (since the terminal normally sends three or six word transmissions), there exists a single output message object that all processors share. Each contributes a delta update until the message is long enough for effective transmission.

Various locking mechanisms, described above, make this sharing possible through a crude form of synchronization. Shared memory resembles a blackboard structure when multiple processors are involved, because each scans shared memory to determine what to do.

For example, an output process constantly looks for messages with status "output." (There is one output process for each direction (to terminal, to host).) Two antecedents are enough. In the following case, the right output processor selects its messages:

IF	The status of ANY message is output
	The direction of THAT message is right
THEN	LOCK that message
	THAT message is_a output_message
	DEINstantiate that message FROM temporary_message

R. META CONTROL AND META LANGUAGE

As will be shown in the detailed descriptions below, the meta language is generally rather simple in implementation. Agent has been designed to minimize the complexity of the meta language needed to drive it.

The metalanguage is interpreted and is loaded into Agent at initialization time.

A small run-time kernel for each process is needed to interpret and execute the meta rules for that process.

1. Meta Control Functions

a. *LOOP*

(LOOP (<fact>) (<factset>) (<scope>))

Repeatedly executes the scope as long as the fact is true in the factset.

b. *EVENT*

(EVENT (<fact>) (<scope>))

EVENT executes the scope whenever the fact is true in the default or global factset. This is a way that system events (such as “shut down”) can affect running processes. EVENT differs from LOOP, in that EVENT is triggered by “true” condition in the default, or global factset. LOOP is a directed (program) statement, which will execute when directed, and continue execution as long as the determining fact is true.

c. *INFER*

(INFER (<knowledge base>) (<input facts>) (<output facts>))

INFER causes the knowledge base specified to be processed using the input facts. The result of the inferencing process is the set of output facts.

d. *EXIT*

(EXIT) causes the process to terminate.

e. *CREATE*

(CREATE (<factset>) (<factsetname>))

CREATE explicitly creates a factset with the name supplied. Factsets are also created implicitly by reference.

f. RELEASE

(RELEASE (<factset>))

This deletes the named factset.

g. ADDFACT

(ADDFACT (<fact>) (<factset>))

Adds a fact to a factset.

h. DELFACT

(DELFACT (<fact>) (<factset>))

Removes a fact from a factset.

i. IS_TRUE

(IS_TRUE (<fact>) (<factset>))

Returns TRUE if the fact is true in the factset.

j. IS_FALSE

(IS_FALSE (<fact>) (<factset>))

Opposite of TRUE.

k. IF

(IF (<truth function>) (<scope>))

IF executes the scope when the truth function evaluates to true. Generally, this will be seen as follows:

(IF (TRUE (<fact>) (<factset>)) (<scope>))

l. MERGE

(MERGE (<factset1>) (<factset2>) (<factset3>))

Causes the facts from factset1 to be combined with the facts in factset2 to form factset3 (which can be the same name as either of the first two sets). If a fact is true in either factset1 or factset2 then the true fact is in the output factset3. If both factset1 and factset2 are true, both truths are added to the output factset3. If neither factset1 or factset2 are true, then this instance of the merge produces a null output, or no merge for this fact only.

m. EFFECT

(EFFECT (<function>) (<parameter list>))

EFFECT calls the function given with the parameter list given. This is a catch-all for any function that is useful.

n. NOTIFY

(NOTIFY (<string>))

Sends a string to the operator console.

IV. AGENT ARCHITECTURE

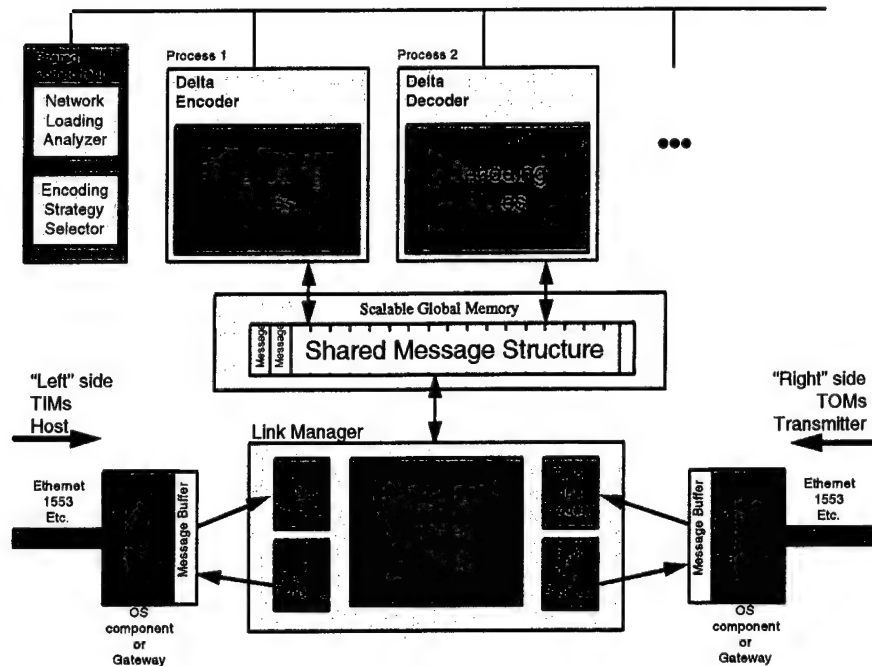


Figure 20, Agent Architecture

A. LINK MANAGER

The Link Manager enforces the time rules of the network, as well as performing housekeeping functions for Agent. If only the Shared Message Structure and the Link Manager existed, the network into which Agent was inserted would function normally. Messages would be intercepted from the left (host) or right (terminal), be placed into the structure, age, and then be asserted on the output network or bus in the correct direction (that is, to the host or to the terminal).

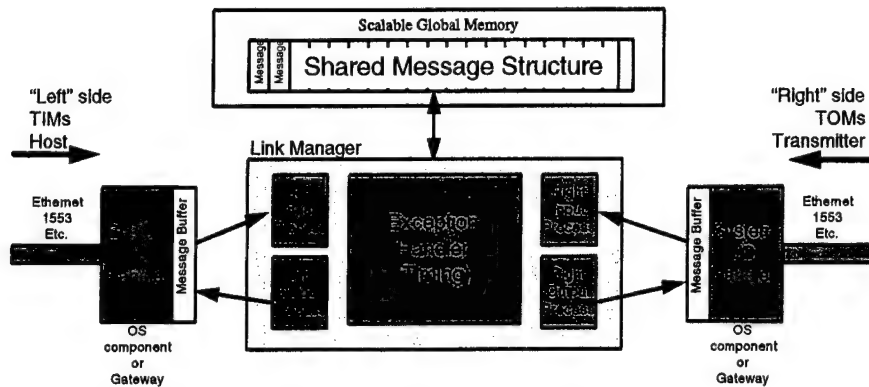


Figure 21, Link Manager

Agent, at the physical level, is a severance implementation. That is, the network or bus between the host and the terminal are physically severed, and the Agent hardware is inserted at the severance point. The software functions by interception. Messages coming in either direction are intercepted by Agent, processed, and then asserted back on the network or bus. The messages may be asserted in their original form or in Agent format. If repetition rate reduction is in effect, outgoing messages (that is, messages from the host to the terminal) can be deleted by Agent.

Each of the components of the Link Manager is a separate process. They may very well be implemented on separate processors, in order to meet run time requirements. They all operate asynchronously, communicating indirectly through the Shared Message Structure.

B. SIMPLIFYING ASSUMPTIONS FOR THE PROTOTYPE

It is assumed that J-Series messages of types 3.2 (air track), 3.3 (surface track), and 2.2 (Air PPLI) are of interest—all others are not. This assumption, as, indeed, all other assumptions, is entirely rule or fact driven and not internal to Agent.

It is assumed that there is a fixed-length message header in all outgoing messages. It is assumed that the type of message is in this header as well as in the message itself.

The time the message was originated is presumed to be in the header. There is also a field in the header that indicates that an outgoing message must be acknowledged. However, this field is ignored at this time because none of the types processed require acknowledgment. If a type is at some point handled that does require acknowledgment, it can simply be passed through as is (it is presumed that such messages are a rare occurrence).

For simplification, each J-Series message is presumed to be the same length. Each contains the following fixed-length fields:

- Message Type
- Track Number
- Update Time
- Latitude
- Longitude

Of course, real messages have considerably more fields than this. However, they would be handled in precisely the same way as the identified fields. It is presumed that the field positions within the three update types of interest are identical.

The "Update Time" field refers to the time of the observation or update, not the time the message was originated or the current time. It is also presumed that the time in a series of these updates is predictable plus or minus a small delta. Therefore an Agent, looking at the last few update messages, can determine if it has a complete recent sequence from these times.

1. Left Input Process

The left input process is alerted to the arrival from the host of a message. It has the following responsibilities:

- Receives event notification from system I/O handler (left side)
- Determines message type
- Instantiates the message in shared memory
- If of an interesting type (e.g., one that will eventually be transmitted in raw or

Agent

form by the terminal) it sends it to the Message Store

- Sets the direction to "left"

Sets the status slot to "entry"

a. Production Rules

This knowledge base is identified as "LeftInput." It requires no facts except the event type (it doesn't actually need the event type either, if the only event type it handles is new message).

```
IF          Event is new_message
THEN        LOCK n7
            n = n + 1
            UNLOCK n
            message[n] is_a message8
            &nchars = 0
            The string of message[n] is FIELD
              (left_input_buffer,1,&nchars)9
            Type of message[n] is FIELD (string of message[n],
              position of type, size of type)
ELSE        EXIT
```

⁷ There are three processes that can create new messages, and all must use unique names to do so. Therefore they all use the global slot "n" to do this. If the variable is already locked the process hangs until the variable is unlocked by the locking process.

⁸ Until we know what type of message we're handling, it will be instantiated simply as a "message." Later, it will be instantiated properly (using multiple instantiation), then deinstantiated from message (so that we won't have multiple links to the same message in the same subtree of the semantic network.

⁹When the third argument to FIELD is null or zero, this is a signal for FIELD to return the number of characters transferred from a system routine. When the third argument is positive, it specifies the number of characters FIELD will return. In this case we are interested in the entire message, so nchars is set to zero before the call, insuring that all of the input data will be transferred.

IF	The FIELD (type of message[n], 1, 1) is "J" ¹⁰
THEN	The class of message[n] is interesting ONLY
ELSE	The class of message[n] is uninteresting ONLY
	The type of message[n] is housekeeping
	Message[n] is_a output_message
	The direction of message[n] is right
	The status of message[n] is output
	DEINstantiate ¹¹ message[n] FROM message ¹²
IF	(The type of message[n] is J3.2 OR
	The type of message[n] is J3.3 OR
	The type of message[n] is J2.2)
THEN	The interest of message[n] is targeted ONLY
ELSE	The interest of message[n] is not_targeted ONLY
IF	The interest of message[n] is targeted
THEN	Message[n] is_a track_update ¹³
	DEINstantiate message[n] FROM message
	The message_time ¹⁴ of message[n] is FIELD (string of
	message[n], position of message_time, size of
	message_time)
	The update_time ¹⁵ of message[n] is FIELD (string of
	message[n], start of time, size of update_time)
	The track of message[n] is FIELD (string of message[n],
	position_of_track, size of track)
	The lat of message[n] is FIELD (string of message[n], start
	of latitude, size of latitude)
	The lon of message[n] is FIELD (string of message[n],
	start of longitude, size of longitude)
	The header of message[n] is FIELD (string of
	message[n],1, size of header)

¹⁰ Using quote marks ensures that the comparison will be to the character J. Otherwise the comparison would be to the value found in the J slot of the null frame.

¹¹ Recall that this DEINstantiate removes the message only from the class message. It has also been instantiated as a track update message and is not removed from class track update. IF there had not been multiple instantiations, the message would have been deleted by DEINstantiate.

¹² Note that we did not instantiate this message as a "housekeeping message" because, unless the actual implementation gets into buffer management, housekeeping messages mean nothing to Agent

¹³ Although we could retrieve the message type using FIELD, we already know it has to be "update" because of the J-series type.

¹⁴ The "message time" is the time in the message header used for host-terminal bookkeeping and checking.

¹⁵ The "update time" is the time of the sighting, as produced by surveillance processing. It is separate and distinct from the message time. "At update time X vehicle Y was at position Z, and the message was created/sent/originated at message time A" may demonstrate the difference in meaning.

The direction of message[n] is left
 LOCK J¹⁶
 J = J + 1
 UNLOCK J
 message[j]¹⁷ is_a raw_stream
 The size of message[j] is &nchars - headersize_out¹⁸
 The message_time of message[j] is the message_time of
 message[n]
 The status of message[n] is entry¹⁹
 EXIT

IF The class of message[n] is interesting
 The value of message[n] is not_targeted
 THEN Message[n] is_a output_message
 DEINstantiate message[n] FROM message
 The direction of message[n] is right
 LOCK J
 J = J + 1
 UNLOCK J
 message[j] is_a raw_stream
 The size of message[j] is &nchars - headersize_out
 The message_time of message[j] is the message_time of
 message[n]
 The status of message[n] is output

b. Meta Rules

```

(Loop (Event20 ("Left Input Message"))
(
    (Infer ("Event is new_message") ("LeftInput")
      ("OutFacts")))
)
  
```

¹⁶ J has to be locked because the right input process and the decoding processes create such objects and all have to be unique. Naturally, all processes that create history records use J.

¹⁷ Case is ignored in Agent. Case can be used in the facts and rules to make them easier to read.

¹⁸ J-Series messages can be of several different lengths. The header is of fixed length for all host-to-terminal messages. If the trigger is supplied by operating system elements or Gateway, and the number of characters in the input buffer can be known, then Agent does not have to keep track of the size of each message type. On the other hand, if the input components do not tell the size of the message, then input facts must supply the length in characters of each formatted message.

¹⁹ All other processes are concerned primarily with the status of a message. Until the status slot is set, no other process will touch the message being built. As soon as that status is set the rule bases of other processes may operate on the message.

²⁰ EVENT waits until triggered by a system process, Gateway, or other process.

2. Left Output Process

The left output process asserts messages on the network connection leading to the host computer system. For Link-16, these are known as TOMs (Terminal Output Messages). Agent messages can also be sent to the host by this process. For other digital networks where a severance implementation is appropriate, these are messages originating from the right side.

a. *Production Rules*

IF	The status of ANY message is output The direction of THAT message is left
THEN	&pos = 1
IF	The message of THAT message is to_be_constructed The type of THAT message is J-Series
THEN	The string of THAT message is NULL The string of THAT message is FIELD (the string of THAT message, &pos, MAKEHEADER()) ²¹ &pos = &pos + size of right_header The FIELD (the string of THAT message, &pos, the track of that message) &pos = &pos + size of track The FIELD (the string of THAT message, &pos, the lat of that message) &pos = &pos + size of lat The FIELD (the string of THAT message, &pos, the lon of that message) &pos = &pos + size of lon The FIELD (the string of THAT message, &pos, the update_time of that message) The status of THAT message is old The message of THAT message is constructed ONLY
ELSE	EXIT "Error—Can only construct update messages"
IF	The message of THAT message is constructed

²¹ In Link-16, there is a five word header on TOMs, or Terminal Output Messages. MAKEHEADER creates this legal header. Since the contents of the headers of both TIMs and TOMs is classified, this function, or, most probably, knowledge base, has been omitted.

THEN	The type of THAT message is <code>_not J-Series</code> ²² FIELD (left_output_buffer, 1, size of THAT message) ²³ TRIGGER (left_output_buffer, &pos) ²⁴ EXIT
IF THEN	The message of THAT message is constructed LOCK J J = J + 1 UNLOCK J message[j] is <code>_a raw_stream</code> The size of raw_stream[j] is SIZE (string of THAT message) ²⁵ - headersize_in ²⁶ The message_time of raw_stream[j] is the message_time of THAT message FIELD (left_output_buffer, 1, size of THAT message) ²⁷ TRIGGER (left_output_buffer, &pos) ²⁸

3. Exception Handler

The exception handler has several responsibilities. First, it must see that messages that have been delayed too long by Agent (too long as defined in the facts) are properly

²² Therefore it has to be an housekeeping message.

²³ The FIELD function now moves the message string, which will look to the host as if it were a standard message, to the system (or Gateway) output buffer.

²⁴ TRIGGER tells the system that the buffer specified by the argument is ready for insertion on the bus or network connection on either side of Agent. The second argument is the size of the message. These arguments are notional in that the specific steps and forms will be determined for Agent during detailed design for a particular installation.

²⁵ SIZE returns the number of characters found in its argument. SIZE must be used carefully where values are subject to concatenation. In general, if there are multiple values, SIZE returns the length of only the first value.

²⁶ The header of an "in" message refers to the header of a message from terminal to host. The header of an "out" message refers to the header of a message going to the terminal. J-Series messages can be of several different lengths. The header is of fixed length for all host-to-terminal messages (currently 10 words) and terminal to host message (currently 5 words). If the trigger is supplied by operating system elements or Gateway, and the number of characters in the input buffer can be known, then Agent does not have to keep track of the size of each message type. On the other hand, if the input components do not tell the size of the message, then input facts must supply the length in characters of each formatted message.

²⁷ The FIELD function now moves the message string, which will look to the host as if it were a standard message, to the system (or Gateway) output buffer.

²⁸ TRIGGER tells the system that the buffer specified by the argument is ready for insertion on the bus or network connection on either side of Agent. The second argument is the size of the message. These arguments are notional in that the specific steps and forms will be determined for Agent during detailed design for a particular installation.

forwarded within allowable time limits. Second, it looks for "collisions," which occur when an update on a track is received before a pending update has reached output status. A collision forces the pending message to be sent immediately (with all of the other messages thus far batched with it). Finally, it "prunes" the shared message structure of message so old as to no longer be of interest.

a. *Production Rules*

Rules for the exception handler are central to the overall design of the rule bases.

The first is a message whose status has stayed "entry" too long (the "Aged Entry Message"). Because its status is "entry" it is not either now or soon to become a component of the temporary_message or the output_message. Therefore it is instantiated as an output_message and otherwise deinstantiated.

If the status of a message is "encoding," then some process has either already added it to the temporary_message or will soon do so. The exception handler will output the temporary message if the ID is there. Otherwise it does nothing at this point because one of the processes will soon add the ID to the temporary message. If the entry is idle for too long a period of time, as determined by network rules of operation, the message, upon updating of the temporary storage buffer, will have the appropriate ID established, and transmitted. This is not an arbitrary process, but rather a trade off between the desire to maximize the fill of each transmitted packet, v.s. the time constraints of transmission. At a subsequent inferencing, that rule base will find the ID in the temporary message and force that message to be sent.

The third exception has to do with an update arriving for the same track that is currently being processed. In this version of the rules, the base forces the output of the temporary message and allow the new update to be processed normally.

Old messages that are no longer of value to Agent are pruned so that the space can be made available for additional messages. This store of old message is used by various heuristics to construct or deconstruct Agent reduced messages.

Finally, objects which are used to determine network and NPG loading are purged after they are no longer useful in determining net and node saturation levels.

(1) Aged Entry Message

IF The status of ANY message is entry²⁹
 The direction of THAT message is left
 The type of THAT message is track_update
 The TIME_f³⁰ minus the message_time of THAT message
 is_greater_than the update_processing_limit
THEN LOCK THAT message
 The status of THAT message is output
 The direction of THAT message is right
 The status of THAT message is old
 The message of THAT message is constructed ONLY
 UNLOCK THAT message

IF The status of ANY message is entry
 The direction of THAT message is right
 The type of THAT message is track_update
 The TIME_f minus the time of THAT message
 is_greater_than the update_processing_limit
THEN LOCK THAT message
 The status of THAT message is output
 The direction of THAT message is left
 The status of THAT message is old
 The message of THAT message is constructed ONLY
 UNLOCK THAT message

IF The status of ANY message is entry
 The direction of THAT message is right
 The type of THAT message is housekeeping
 The TIME_f minus the time of THAT message
 is_greater_than the housekeeping_processing_limit
THEN LOCK THAT message
 The status of THAT message is output
 The direction of THAT message is left
 The status of THAT message is old
 The message of THAT message is constructed ONLY
 UNLOCK THAT message

IF The status of ANY message is entry
 The direction of THAT message is right
 The TIME_f minus the time of THAT message

²⁹ These rules cover the problem of an entry message (left or right) that has sat too long without being processed (encoded or decoded).

³⁰ This is the current system time (or other appropriate time reference).

THEN is_greater_than the general_processing_limit³¹
 LOCK THAT message
 The status of THAT message is output
 The direction of THAT message is left
 The status of THAT message is old
 The message of THAT message is constructed ONLY
 UNLOCK THAT message

(2) Aged message being encoded³²
 IF The status of ANY message is encoding^{33,34}
 The TIME_f minus the time of THAT message
 is_greater_than the general_processing_limit³⁵
 THEN LOCK THAT message
 LOCK the temporary_message
 The status of THAT message is late

IF The status of ANY message is late
 The track of THAT message is_the_same_as the track of
 the temporary message
 THEN LOCK shared memory
 The status of THAT message is old
 The temporary_message is_a output_message
 gpos = 0³⁶
 DEINstantiate THAT message from

³¹ The processing time limits have not been determined. It is probable that all J-Series messages will have the same time limit, and, perhaps, the housekeeping messages as well. On the other hand, they could be different. If so, each message type may have a time limit associated with it and these rules must be expanded accordingly.

³² Note that aging messages being decoded cannot be handled as exceptions because there is nothing Agent can do but continue to decode them. Until it decodes the message it cannot even be aware of which tracks are involved, and therefore cannot sense track collision (new update arriving before old update is processed). On the other hand, decoding is one of the fastest processes. The processor must be specified so that all messages are decoded fast enough so as not to be considered late.

³³ These rules handle the case where it is taking too long to batch various messages for output. What will happen is that the temporary message, which is where all processes assemble batched messages, will be output as is and the original messages that are part of that temporary messages will be deleted from shared memory. This will result in a message with fewer updates than is possible, but ends any further delays.

³⁴ If the status is "encoding," and it is true that the message originated from the left (host) side, and Agent is presently operating with reduction, compression, or both. Since all reduced updates must be batched, the results of this encoding appear in the temporary message.

³⁵ Additional rules may be needed if the processing times are different, as explained in a previous footnote.

³⁶ Gpos is a global counter used by the encoding heuristics to mark the next available place in the string of the Agent message being constructed. When a message is deinstantiated from the temporary message class, then this counter must be reset. Since the whole of shared memory is locked at this point, no special lock is required.

```

                                temporary_message
                                UNLOCK shared memory
                                UNLOCK THAT message

IF      The status of ANY message is late
        The track of THAT message is _not the track of the
                                temporary_message
THEN    CONTINUE37

(3)    Collision
IF      The status of ANY message is entry
        The track of THAT message is _the_same_ as the track of
                                the temporary_message
        The direction of THAT message is left
THEN    LOCK THAT message
        LOCK the temporary message38
        The direction of the temporary message is right
        The temporary_message is _a output_message
        The status of THAT message is old
        gpos = 0
        UNLOCK THAT message
        DEINstantiate the temporary_message

IF      The status of ANY message is entry
THEN    The &track is the track of THAT message

IF      The status of ANY message is encoding
        The track of THAT message is the track of the
                                temporary_message39
        The track of THAT message is &track40
THEN    LOCK the temporary message41
        The direction of the temporary message is right
        The temporary_message is _a output_message
        gpos = 0

```

³⁷ If the track ID isn't in the temporary message now, it will be shortly. The problem will be caught on a subsequent iteration of the rules.

³⁸ The temporary message class is only used for messages being encoded. Therefore the direction will always be right.

³⁹ Added last deliberately for the purpose of having this test work.

⁴⁰ We're looking here for an update to a track that is already being encoded. We can detect this because the track number is in the temporary message. This means that the track number is also in a message labelled encoding. And a new update has arrived. Therefore the action is to force out the temporary message before processing the new update.

⁴¹ The temporary message class is only used for messages being encoded. Therefore the direction will always be right.

The status of THAT message is old
 UNLOCK THAT message
 DEINstantiate the temporary_message

(4) Message Pruning

IF The status of ANY message is old
 THAT message is_a track_update⁴²
 The NUMBEROF⁴³(THOSE messages) is_greater_than
 one⁴⁴
 The TIME_f of ANY message minus the time of THAT
 message is_greater_than the persistence_limit⁴⁵
 THEN DEINstantiate THAT message⁴⁶

(5) History Record Pruning

IF If the TIME_f minus the time of ANY history_record
 is_greater_than 2 * net_frametime
 THEN DEINstantiate THAT history_record

b. Exception Handler Meta Rules

Metarules are designed to be simple, and, in the case of the Exception Handler, they are very simple indeed. There is one knowledge base, called "exceptions," detailed in the section above. All the meta rules do is loop on this knowledge base.

```
(LOOP ()
(
(INFER () ("exceptions") ("OutFacts"))
)
```

This data base is interesting because it requires no input fact list (it merely needs access to Shared Memory), and that it has no use for the output facts it produces. The fact that the input FactSet is null means that each time it is executed the old facts generated through inferencing are gone. The purpose of the data base, in this instance, is to provide the unique knowledge reservoir of the associated network(s) principles of operation. In

⁴² Other types of messages (e.g., housekeeping, Agent) are pruned when decoded or encoded.

⁴³ NUMBER returns the number of objects instantiated to the class mentioned.

⁴⁴ The first ten numbers are defined as their value in the null class for readability.

⁴⁵ The persistence limit is how long we need to keep old messages before they no longer have value. At this point a function could be added to archive the messages about to be deinstantiated.

⁴⁶ This is an example of subsetting. The set of messages identified by the first use of ANY is further subsetting by the second ANY, so that the DEINstantiate directive only applies to the smallest subset

this way the Exception Handler can be designed in such a manner as to be independent of specifics, with respect to a particular network's rules of operation, while providing unique capability in each instance.

Of course, other actions can be added, such as logging functions, as desired.

c. Right Input Process

The right input process accepts messages from the terminal or right side. If they are standard messages they are simply marked for output after keeping track of the implied network loading. If they are Agent message they are marked for decoding after keeping track of the amount of data (bits) that were transmitted over the RF network.

(1) Production Rules

The name of this knowledge base is "InputRight."

IF	Event is new_message
THEN	LOCK m
	m = m + 1
	UNLOCK m
	message[m] is_a message
	The type of message[m] is FIELD (type_position of
	right_header, size_of_type of right_header)
IF	The type of message[m] is housekeeping
THEN	The string of message[m] is FIELD (right_input_buffer, 1,
	update_size)
	The direction of message[m] is left
	The message of message[m] is constructed
	The form of message[m] is raw
	The status of message[m] is output
	EXIT
IF	The FIELD(type of message[m],1,1) is "J"
THEN	The type of message[m] is J-Series ⁴⁷
IF	(The type of message[m] is J3.2 OR
	The type of message[m] is J3.3 OR
	The type of message[m] is J2.2 OR)
THEN	Message is interesting ONLY

⁴⁷ Recall that slot assignment implies concatenation for non-numeric values. Thus the message type is both "J-Series" and its original type (e.g., J3.2, J3.3, or J2.2).

ELSE	<p>Message is uninteresting ONLY</p> <p>The form of message[m] is raw</p>
IF THEN	<p>The type of message[m] is Agent</p> <p>Message is interesting ONLY</p> <p>The form of message[m] is encoded</p>
IF THEN	<p>Message is uninteresting</p> <p>The direction of message[m] is left</p> <p>LOCK J</p> <p>J = J + 1</p> <p>UNLOCK J</p> <p>message[j] is_a raw_stream</p> <p>The rawsize of message[m] is (SIZEOF (string of message[m]) - headersize_out)</p> <p>The time of message[m] is FIELD (string of message[m], start of update_time, length of update_time)</p> <p>The message of message[m] is constructed</p> <p>The form of message[m] is raw</p> <p>Status of message[m] is output</p> <p>EXIT</p>
IF THEN	<p>Message is interesting</p> <p>Type of message[m] is Agent</p> <p>Message[m] is_a Agent_message</p> <p>DEINstantiate message[m] FROM message</p> <p>The string of message[m] is FIELD (right_input_buffer, 1, update_size)</p> <p>The encoding_scheme⁴⁸ of message[m] is FIELD (right_input_buffer, position of scheme, size of scheme)</p> <p>The message_time of message[m] is FIELD (right_input_buffer, position of message_time, size of message_time)</p> <p>The direction of message[m] is right</p> <p>The form of message[m] is encoded</p> <p>The status of message[m] is entry</p> <p>LOCK J</p> <p>J = J + 1</p> <p>UNLOCK J</p>

⁴⁸ The Agent encoding scheme appears either in the message header, if possible, or in the encoded message itself. If the latter, then the first part of the message information is not incoded. This enables the receiving node to decode Agent-encoded messages without have to be synchronized with the sending node as to what scheme is being employed.

```

        message[j] is_a encoded_stream
        rawsize of message[j] is SIZEOF (string of message[m]) -
            header_in
        time of message[j] is message_time of message[m]
        EXIT

    IF      Message is interesting
    THEN    Type of message[m] is J-Series
            The type of message[m] is_a track_update
            DEINstantiate message[m] FROM message
            The string of message[m] is FIELD (right_input_buffer, 1,
                update_size49)
            The message_time of message[m] is FIELD
                (right_input_buffer, position of message_time, size
                of message_time)
            The track of message[m] is FIELD (right_input_buffer,
                position_of_track, size of track)
            The lat of message[m] is FIELD (right_input_buffer, start
                of latitude, size of latitude)
            The lon of message[m] is FIELD (right_input_buffer, start
                of longitude, size of longitude)
            The update_time of message[m] is FIELD
                (right_input_buffer, start of update_time, size of
                update_time)
            The header of message[m] is FIELD (right_input_buffer,
                start of header, size of header)
            The direction of message[m] is right
            The status of message[m] is output
            The form of message[m] is raw
            LOCK J
            J = J + 1
            UNLOCK J
            message[j] is_a raw_stream
            rawsize of message[j] is SIZEOF (string of message[m]) -
                header_in
            time of message[j] is message_time of message[m]
            EXIT

```

(2) Meta Rules

```

(LLOOP (EVENT ("Right input message"))
(

```

⁴⁹ This assumes that all three update types of interest are the same length. If they are of different lengths then there has to be a rule for each message type and a specific update size assigned to each type.

(INFER ("Event is new_message") ("InputRight")
("OutFacts"))

)

d. Right Output Process

The right output process takes messages of any class whose status is "output" and whose direction is "right" and asserts them on the bus or network to the terminal. It also keeps instantiates the necessary loading measurement objects.

With respect to loading, left input process has kept track of the size of raw messages from the host. The left output process keeps track of the size of raw messages from the terminal. This process keeps track of the size of messages as they will actually be transmitted.

When an Agent message is decoded, the left output process will kept track of the size of messages as received (and, therefore, take credit for the size reduction of Agent operations). When messages are encoded, the size of the unencoded versions is recorded by the encoding heuristics in the temporary message. Therefore, when it gets instantiated as an output message, the size that would have been transmitted is known as well as the size actually transmitted.

(1) Production Rules

This knowledge base is called "rightout."

IF	The status of ANY message is output The direction of THAT message is right The class of THAT message is interesting (The form of THAT message is raw OR The message of THAT message is constructed) ⁵⁰
THEN	FIELD (string of THAT message, position of message_time, size of message_time) is TIME _f ⁵¹ FIELD (right_output_buffer, 1, SIZEOF(THAT message)) TRIGGER (right_output_buffer, &pos) ⁵²

⁵⁰ This message already has a header

⁵¹ Time if a function of message input output, plus processing differential.

⁵² TRIGGER tells the system that the buffer specified by the argument is ready for insertion on the bus or network connection on either side of Agent. The second argument is the size of the message. These arguments are notional in that the specific steps and forms will be determined for Agent during detailed design for a particular installation.

```

The status of THAT message is OLD
LOCK J
J = J + 1
UNLOCK J
message[j] is_a raw_stream
rawsize of message[j] is SIZEOF (string of THAT
    message) - header_in
time of message[j] is message_time of THAT message

IF      The status of ANY message is output
        The direction of THAT message is right
        The class of THAT message is interesting
        The form of THAT message is encoded
THEN    FIELD (string of THAT message, position of
            message_time, size of message_time) is TIME53
        FIELD (right_output_buffer, 1, size of THAT message)
        TRIGGER (right_output_buffer, &pos)
        The status of THAT message is OLD
        LOCK J
        J = J + 1
        UNLOCK J
        message[j] is_a encoded_stream
        rawsize of message[j] is SIZEOF (string of message[m]) -
            header_in
        time of message[j] is message_time of message[m]
            object[&m] is_a encoded_stream
        The exploded_size of message[j]]54 is the exploded_size of
            THAT message

```

(2) Meta Rules

```

(LLOOP (FALSE(EVENT ("terminate"))))
(
    (INFER () ("rightout") ("OutFacts"))
)

```

⁵³ Since this is an encoded message, Agent created it from one or more other formatted messages. Or it could be an internal Agent-to-Agent message. In any case, it must be assigned a message time that the terminal will consider to be legal. In this example, the current system time is used as the message time. During detailed design the timing problems will be fully resolved.

⁵⁴ The "exploded size" is simply the size of the messages that would have had to have been transmitted had Agent not been operating. This value is calculated as the temporary message is constructed, update by update.

C. DELTA COMPONENTS

There are two processes (minimum) of the delta components—an encoding process and a decoding process. They operate asynchronously of each other. There must be at least one of each but there can be as many of each as needed to keep up with the loading. Thus a platform with no reporting responsibilities within the Surveillance NPG may have a single encoder process and as many decoders as are needed to keep up with the flow of messages. A node with reporting responsibilities may have additional encoders in order to keep up with the encoding load. A multiprocessor environment is best for this type of architecture. Figure 22 shows the parts of the architecture discussed below.

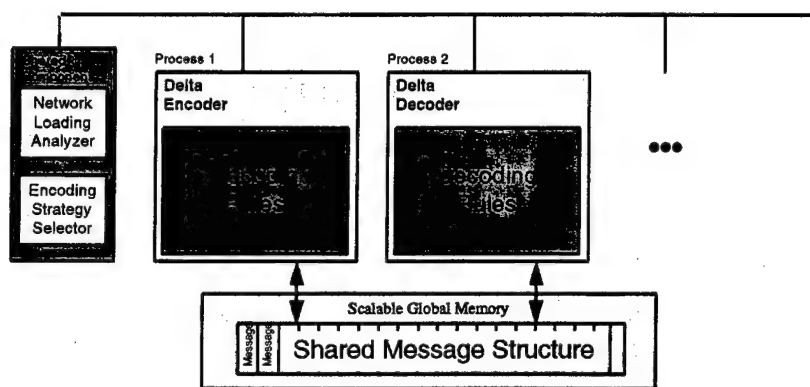


Figure 22, Delta Components and the Shared Component

The creation of new encoding/decoding components is performed by the load analyzer, which has rules to determine the optimal number of components for the current situation.

1. Encoder

a. "Simple" Delta Messaging

The original and simplest form of delta messaging transmits only the fields that have changed from the previous update, and transmits them in their original form. Thus, if only the position and message time have changed, only these two fields will be transmitted. The agent on the receiving side will find the previous message, supply the missing fields, and pass it on to the host.

The complication is that the terminal always sends out three or six word messages regardless of how much data is actually put into the message. Therefore updates must be "batched" as well, so that each Agent message will be transformed into many regular (or "raw") update messages on the receiving end.

If multiple processors are working simultaneously, therefore, they work to build the same update message (which is why, as each update is added, the so-called "temporary message" is locked).

(1) Production Rules

This rule base is called "deltaencode."

IF	The mode of Agent is simple_delta The status of ANY message is entry The direction of THAT message is left
THEN	LOCK_ONE of THOSE messages ⁵⁵ The encode-message of THAT message is NAME (THAT message) ⁵⁶ The status of THAT encode_message is encoding UNLOCK THAT message
IF	The track of ANY message is _the_same_as the track of

⁵⁵ LOCK ONE is a special version of LOCK. Only one of the messages of the set identified in the antecedents is locked. Which one is locked is entirely arbitrary. It is assumed that the implementation of the semantic network will cause the eldest message marked "entry" to be selected. Just as nested specifiers reduce the set referred to by THAT or THOSE, so does LOCK. After LOCK, THAT or THOSE refer only to the locked objects—in this case the message we wish to handle.

⁵⁶ NAME is a function that returns the name of the object. The name was given by concatenation when the object was first instantiated. Thus subsequent rules can refer to that message by name.

```

                                $encode_message57
                                The type of THOSE messages is old
                                The message_time of the $encode_message minus the
                                  message time of ANY of THOSE messages
                                  is_less_than_or_equal_to the update_frequency of
                                  the $encode_message58 + time_delta59
THEN      The old_encode_message is "empty" ONLY60
                                The old_encode_message is the NAME (THAT message)
                                  ONLY

                                IF
THEN      The old_encode_message is "empty"
                                The direction of the $encode_message is right
                                The $encode_message is_a output_message
                                The status of $encode_message is output
                                EXIT

                                IF
THEN      NUMBEROF (temporary_message) is_greater_than zero
                                CONTINUE61
ELSE      LOCK temporary_message
                                LOCK e62
                                e = e + 1
                                UNLOCK e
                                rmessage[e] is_a temporary_message
                                gpos = 1
                                FIELD (string of temporary_message,gpos,) is the code
                                  for63 simple_delta
                                gpos = gpos + size of simple_delta_code

                                IF
THEN      TRUE
                                FIELD (string of temporary_message, gpos,) is "XX"64

```

⁵⁷ Note that the dollar sign, or evaluation symbol, tells the engine to substitute the contents of encode_message in the antecedent before evaluating further. Since the name of a message is stored in "encode_message," the name of that message is used in the evaluation.

⁵⁸ This value is assigned when the semantic network is built as a slot associated with the message type. Thus a J3.2 may have one update rate and a J3.3 have another. Therefore new targeted messages are assigned to the appropriate type of message rather than, simply, to "message."

⁵⁹ If update times do not form a completely accurate sequence, then perhaps a small delta will be needed to make sure that the correct earlier message was located. If no previous update is found the message will simply be transmitted in raw format.

⁶⁰ Agent cannot use simple delta encoding on a message for which it has no immediate predecessor.

⁶¹ CONTINUE is a no-op.

⁶² Must be global if there are multiple instances of the encoding process operating.

⁶³ "Of" and "for" mean precisely the same.

⁶⁴ XX is an arbitrary field code that indicates that what follows is a new message. The format of the batched message is in general a field code followed by the field itself. Thus field code 1 means update


```

gpos = gpos + 2
FIELD (string of temporary_message, gpos,) is "01"65
gpos = gpos + 2
FIELD (string of temporary_message, gpos, size of type) is
    type of message[e]
FIELD (string of temporary_message, gpos,) is "02"
gpos = gpos + 2
FIELD (string of temporary_message,e) is the track of
    message[e]
gpos = gpos + size of track
FIELD (string of temporary_message,e) is the update_time
    of message[e]66
gpos = gpos + size of update_time

IF      The lat of message[e] is the lat of old_encode
THEN    Continue
ELSE    FIELD (string of temporary_message, gpos, size of lat) is
        the lat of message[e]
        gpos = gpos + size of lat

IF      The lon of message[e] is the lon of old_encode
THEN    Continue
ELSE    FIELD (string of temporary_message, , size of lon) is the
        lon of message[e]
        gpos = gpos + size of lon
        The exploded_size of message[e] is the exploded_size of
            message[e] plus the size of update_message
        UNLOCK temporary_message
        EXIT

```

b. Meta Rules

There is only one set of meta rules for the encode delta component(s). This set is as follows:

time, which all messages have, field code 2 means track number, and so on. The special code of XX means that this is the beginning of a new message, so that the decoding algorithm will start looking for a sequence of field codes and fields for the new message. If no known field code is encountered, then the message end has been found. Since outgoing messages can be of almost any length, modulo three words, Agent keeps adding updates to the temporary word until the exception handler forces it to be output.

⁶⁵ The code for simple delta messaging. In the real implementation, "code of type" should be used. It is given explicitly here to remind the reader that such codes are arbitrary.

⁶⁶ All delta messages begin with the trio of type, track, and update_time. The type is needed to know what fields to look for, the track is needed to match up with previous tracks, and the update time will always be different. Other fields are added only if different from the previous update.

```

(LOOP (FALSE(EVENT ("terminate"))))
(
    (GETFACTS ("Agent Status") (Status_facts))
    (IF (TRUE ("strategy of Agent is simple_delta"))
        (
            (INFER () ("deltaencode") ("OutFacts"))
        ))
    (IF (TRUE ("strategy of Agent is extrapolation"))
        (
            (INFER () ("extrapencode") ("OutFacts"))
        ))
    )
)

```

Any additional encoding schemes can be added in the same fashion to this meta rule base.

2. Decoder

a. *Production Rules—Routing Messages*

IF	If the status of ANY message is entry The direction of THOSE messages is right The form of THOSE messages is encoded
THEN	Action is NULL
ELSE	EXIT
IF	The encoding_scheme of ANY message is simple_delta
THEN	Action is process delta message
IF	The encoding_scheme of ANY message is extrapolation
THEN	Action is process extrapolation ⁶⁷

b. *Production Rules—decoding delta messages*

The following will decode any number of messages encoded with so-called "simple" delta messaging. The name of the base is "decodedelta"

IF	If the status of ANY message is entry The direction of THOSE messages is right The form of THOSE messages is encoded
----	--

⁶⁷ Note that ONLY is not specified in these cases. We are checking to see if messages of either or both schemes are ready for decoding.

```

THEN      LOCK_ONE of THOSE messages
           The status of THAT message is decoding
           &Name is NAME (THAT message) ONLY
ELSE      EXIT68

IF        FIELD (string of $&name, &dpos,2) is "XX"
THEN      &pos = &pos + 2
           LOCK m
           m = m + 1
           UNLOCK m
           message[m] is_a message
           The type of message[m] is FIELD (string of &$name69,
           &pos, size of message_type)
           &pos = &pos + size of message_type

IF        The type of message[m] is J3.2
THEN      message[&m] is_a J3.2
           DEINstantiate message[m] FROM message
           &interval is update_interval of J3.2 ONLY

IF        The type of message[m] is J3.3
THEN      message[&m] is_a J3.3
           DEINstantiate message[m] FROM message
           &interval is update_interval of J3.3 ONLY

IF        The type of message[m] is J2.2
THEN      message[&m] is_a J2.2
           DEINstantiate message[&m] FROM message
           &interval is update_interval of J2.2 ONLY

WHILE     FIELD (string of $&name, &pos, 2) not_equal_to NULL
BEGIN     Decoding

IF        TRUE
THEN      The type of message[m] is FIELD (string of message[&m],
           &pos,2)
           &pos = &pos + 2
           The update_time of message[m] is FIELD (string of
           message[&m], &pos, size of update_time)
           &pos = &pos + size of update_time

```

⁶⁸ If a message to process isn't found that doesn't mean there was an error. A parallel process could have picked up the message for decoding before this process was able to find it.

⁶⁹ Evaluation is from right to left. So that first the parser knows that it is a local variable, second, it knows to perform substitution on it.

```

        The track of message[m] is FIELD (string of message[m],
            &pos, size of track)
        &pos = & pos + size of track

    IF      The track of ANY message is the track of message[m]
        The update_time of THAT message is the update_time of
            message[m] - &interval
    THEN    #old_message = NAME (THAT message)
    ELSE    DEINstantiate message[m]
            EXIT70

    IF      FIELD (string of message, &pos,2) is lat_type
    THEN    &pos = &pos + 2
            Lat of message[m] is FIELD (string of message[m], &pos
                + size of lat, size of lat)
            &pos = &pos + size of lat
    ELSE    Lat of message[m] is lat of #old_message

    IF      FIELD (string of message, &pos,2) is lon_type
    THEN    &pos = &pos + 2
            Lat of message[m] is FIELD (string of message[m], &pos
                + size of lon, size of lon)
            &pos = &pos + size of lon
    ELSE    Lat of message[m] is lat of #old_message

    IF      TRUE
    THEN    Direction of message[m] is left
            Status of message[m] is output
            LOCK J71
            J = J + 1
            UNLOCK J
            Message[j] is_a_raw_stream
            The rawsize of message[j] is size of update
            The time of message[j] is the update_time of message[m]

    END      Decoding

```

⁷⁰ This is an error exit, in that the previous update message for this track could not be found. Therefore we can only drop the message at this point.

⁷¹ We're going to create an history record for each message decoded. This will be used in calculating the effective bandwidth.

c. Meta Rules

The following meta rules for decoding call knowledge base "checking_for_messages" to determine if there are entry messages from the right that need to be decoded, and then calls the knowledge base appropriate for the decoding. Any number of knowledge bases can be included here. Only one is actually documented at this point (simple delta messaging).

```
(LOOP (FALSE(EVENT ("terminate"))))
(
  (INFER () (checking_for_messages) (out_facts))
  (IF (TRUE ("Action is process delta message")
      (out_facts))
    (
      (INFER () ("deltadecode") ())
    )
  )
  (IF (TRUE ("Action is process extrapolation")
      (out_facts))
    (
      (INFER () ("extrapdecode") ("OutFacts"))
    )
  )
)
```

D. SHARED COMPONENTS

This process analyzes net loading and adjusts the strategy of Agent accordingly. Every time a message is sent or received a history record is created for that transmission. If the message was encoded, the raw message size as well as the encoded message size is included.

The first step is to prune the history records of all information older than one frame. The next step is to determine the total number of characters (or bits) that are being transmitted within the NPG. If this number is higher than the upper trigger level (which is set as a result of laboratory experiments), delta messaging kicks in. When it falls below the lower trigger level, delta messaging kicks out. These changes have an effect only on encoding. Whenever an encoded message is received it is decoded regardless of the status of Agent.

As written, Agent kicks in or out based on NPG loading. It is possible to include node loading as well, with an appropriate exchange of TIMs and TOMs to determine the transmission slots available to the node. This would be compared with its output loading to determine if node saturate were either occurring or close to occurring. If so, a reduction strategy would be enacted for this node (the others would continue to operate according to their information about the NPG and themselves). This could include update rate reduction (which would be handled by the left input process, which would simply discard updates at some adjustable rate so that the transmit requirements would match the transmit capability).

V. AGENT METHODS FOR LINK 16

A. INTRODUCTION

For this project, several techniques are under development for reducing 1) the length of frequently-transmitted messages and, 2) the frequency of transmissions. These techniques are lossless in that there is no concomitant reduction in content or user situation understanding. These techniques are under development for initial evaluation and demonstration, to be later verified in the NRaD Systems Integration Facility (SIF). In the SIF, the primary purpose of establishing key benchmarks is to assist in deciding between alternative object-oriented (OO) implementations. Object-oriented technology will be employed as much as practicable in this demonstration. Irrespective of programming style, benchmarks will establish the best possible methods. Therefore, the prototypes may contain a mix of OO and procedural components. The benchmarks will reveal, for each technique, the best possible gain in virtual bandwidth.

Simultaneously, a software agent will be designed to further reduce bandwidth by adding intelligent, distributed link control, as well as to enhance the message length-reduction strategies with such techniques as transmission frequency reduction. As stated earlier, the name for this agent is "Agent." Software agents have been shown to be especially useful for network management in other contexts. The rules of the link will be instantiated in Agent, which, along with an automated transmission planning capability, can provide substantial gains in virtual bandwidth.

B. ATOMIC DATA ELEMENT TRANSMISSION ("DELTA MESSAGES")

In this technique, redundant elements of messages (e.g., track update messages) will be reduced to a minimum. In general, only the elements of a message that have changed since the last update will be transmitted, together with identifying information. The goal is to provide a decrease of one-third to two-thirds bandwidth requirements for track messages, in addition to only transmitting such Delta Messages when new

information deviates outside expected parameters as determined by relative navigation models, see Extrapolation-Driven Updates, below. The effective reductions would make much needed bandwidth available for other purposes, plus increase the links overall track handling capabilities through being able to identify and update a greater number of tracks.

If the OODBMS chosen by other parts of the RTR program becomes the bottleneck in the process, message objects with strong real-time requirements will be cached in RAM, or other strategies developed, to enable the system to keep up with the link requirements.

Figure 23, Physical Implementation, describes how the smart agent will fit into the existing system. Details of the operations of Agent are described in a previous section.

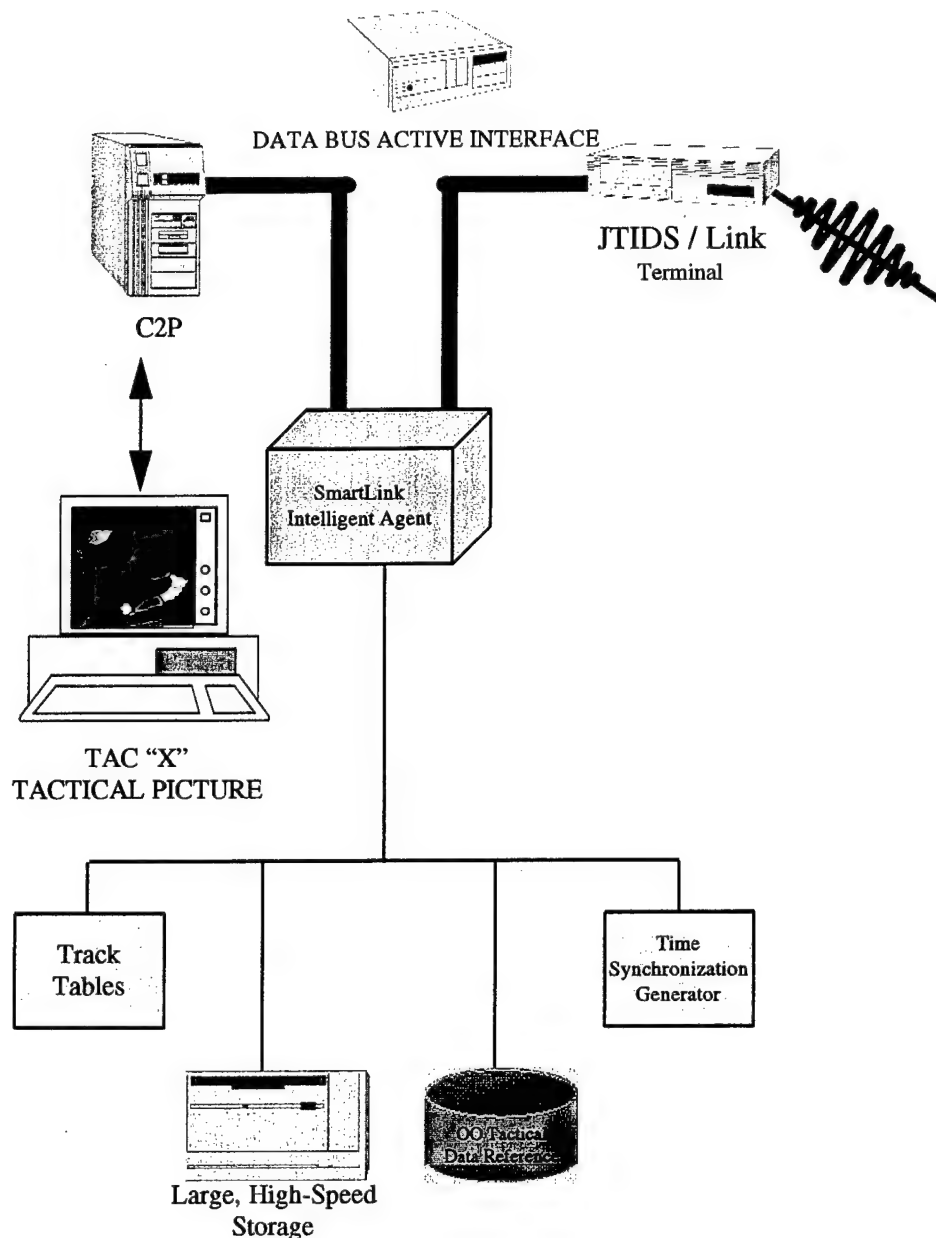


Figure 23, Physical Implementation

Figure 24 shows a conceptual view of delta message reduction. Figure 24 shows the type of information a delta message may contain. Although the TADIL message type is "Free Text," a message type local to Agent will be within the message, for each type of message it transmits. Following the message type is an indicator of the number of fields present for that particular message type. After that is a variable number of fields with field types (i.e. integer, character, binary, etc.) associated with the message type.

Although most fields are fixed-length, some will be variable. For example, with respect to latitude, although a full update message has new data, probably only a very few of the least significant bits from the previous update have changed. Agent will identify and insert the changed bits, and the field length will be the number of them. The receiving Agent will know to replace the lower order bits from the previous message with the new ones. Of course, if the whole field changes, then the whole field will be sent. It is understood that if there is significant change in the original TADIL message, the delta message could be significantly longer than the original message. The implication is that the delta message is checked against the original message size. Only the shorter of the two is sent.

That this kind of severance system can be introduced has already been proven with the Link 16 virtual gateway, which today provides both a passive tap, for relaying the tactical picture to other systems, and the ability to insert messages directly. Agent will build upon the structure of Galaxy.

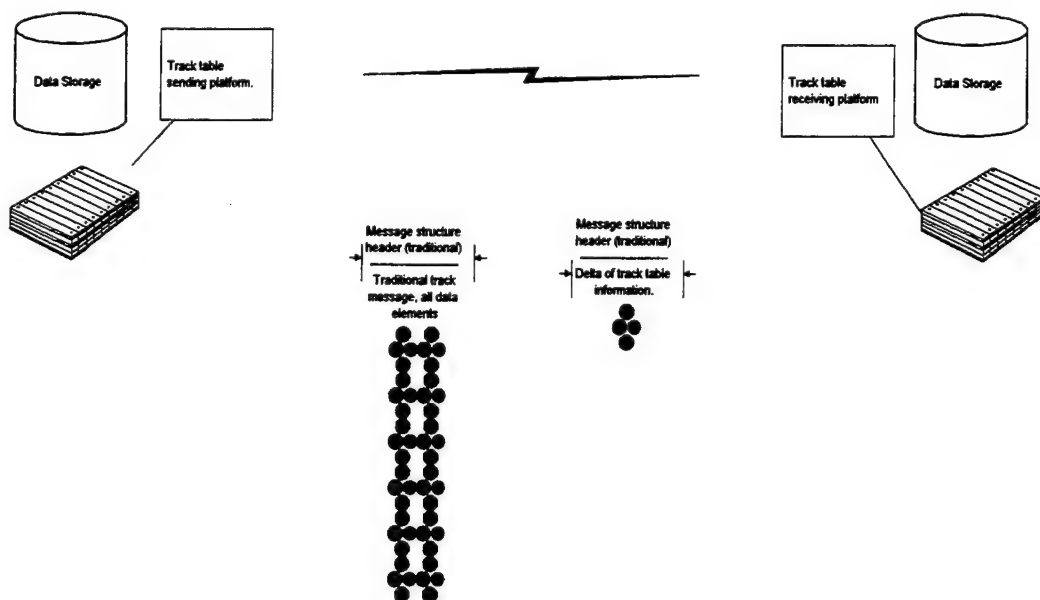


Figure 24, Conceptual Implementation

Free-Text Message

Message Type	Number of Fields	Field Type	Data
Field Type	Field Length	Data	Message 2 Type

Figure 25

Some types of messages internal to Agent will be control messages not directly related to existing message types.

The many different link messages will be surveyed for their applicability to this type of length-reduction technique.

One challenge is that, at present, track update messages belong to the surveillance NPG, which is connected to one of only three existing JTIDS buffers. The buffer is there to ensure that all messages of the surveillance type will be transmitted (in overload conditions, a message not buffered can be tacitly dropped from the system). The C2P manages itself based upon the available buffer size. At present, free-text messages do not belong to the surveillance NPG. This poses no problem for the benchmark demonstrations because there is an operator-assignable buffer that can be used. For the long term, a plan will be created for either 1) including the free-text message in the surveillance NPG, 2) assigning a buffer to the free-text messages, or 3) simulating buffer operations within Agent.

From what is known at the time of this writing, the best answer is to have free-text messages included in the surveillance NPG, which would cause a small modification to the C2P's buffer size prediction algorithms (JTIDS would look more efficient to the C2P than it does today).

C. UPDATE BUNDLING

Not all frequently-transmitted messages are of high priority. Also, the priority of a given type of message may vary with the situation. Delta messages, or other types of messages, can be bundled into a single transmission using the free-text message format. This would make, for example, the transmission of historical track data feasible (by encapsulating a full message and all of the subsequent updates into a single free-text message). Because all of the information is in a single message, rather than in many messages, the system may be able to process the information more efficiently (that is, the total package of information is received more quickly, and less administrative load on the system).

To minimize this overhead, another function might be to "bundle" a historical message with all its subsequent delta messages, so that a platform entering the network could be updated on all of the data concerning a particular track in a single, free-form transmission of minimal size.

In today's system, it is a goal that all track updates be sent within twelve seconds. This is true for slowly-moving objects, such as ships, and faster-moving objects, such as airplanes. Resources permitting, delta messages from slower-moving objects would be "packed" as in figure 4 above, for transmission in a single message after a fixed interval (TBD). At the receiving end, all of the updates would be applied with Agent, with the resulting full update message being sent on to the C2P.

It is important to validate bundling as a concept. This means that in the SIF, the total network loading of sending multiple small messages must be compared to the loading from sending one, larger message. Network performance-measuring tools currently being developed by others will be used to measure the expected gain. Since it is

estimated that network bookkeeping amounts to some twenty percent of net loading, the gain promises to be significant.

This task will be coordinated with task 4.2.5, Active Network Management, in order to implement variable packing. For example, packing parameters can be altered, or packing completely eliminated, as the tactical picture changes.

D. EXTRAPOLATION-DRIVEN UPDATES

In this technique, each type of vehicle being tracked is modeled on every platform, utilizing a flat Earth 3-D model (dead reckoning). Each platform makes a projection of the tracked-vehicle's position at its next-scheduled update cycle, based on observed behavior. If the predicted position is close enough to the actual position, then either no transmission is made and the individual Agents generate the update locally, or a very small transmission is made confirming that the prediction is accurate. If too great a deviation has occurred, then a full update message or a delta message is actually transmitted. Since most tracked vehicles behave predictably over short intervals, this technique promises to greatly reduce network loading—with no loss of information at the receiving platforms.

At any given time, one platform has the reporting responsibility for a given track (based on track quality). The concept of extrapolation-driven updates is to have a set of vehicle simulations at each platform that predict the position of the tracked vehicle at the next scheduled update. If the prediction of the update matches the reported "true" position, within an appropriate tolerance (calculation based upon range and sensor resolution), no update would be sent. Instead, Agent would internally generate an update message based on its prediction and send that update to the C2P. The concept is illustrated in Figure 26.

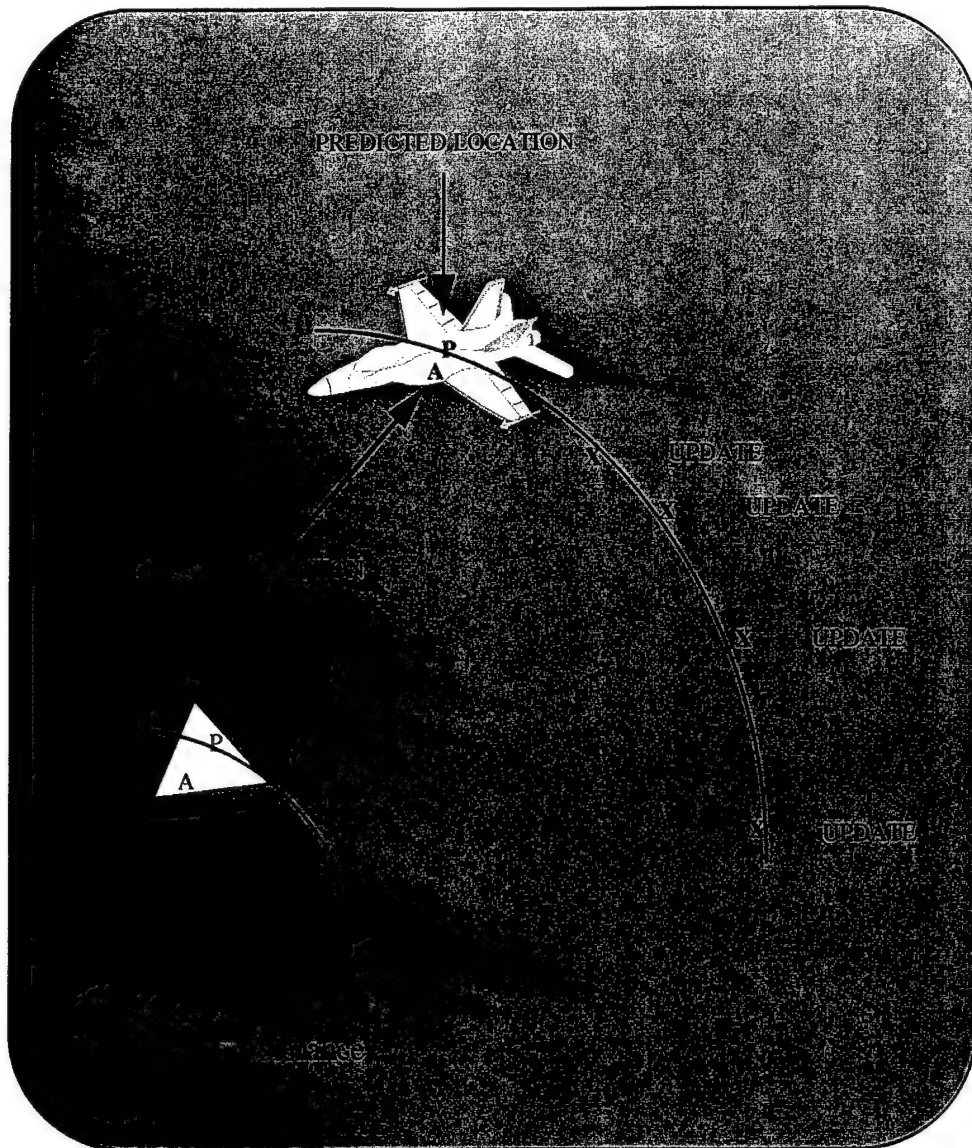
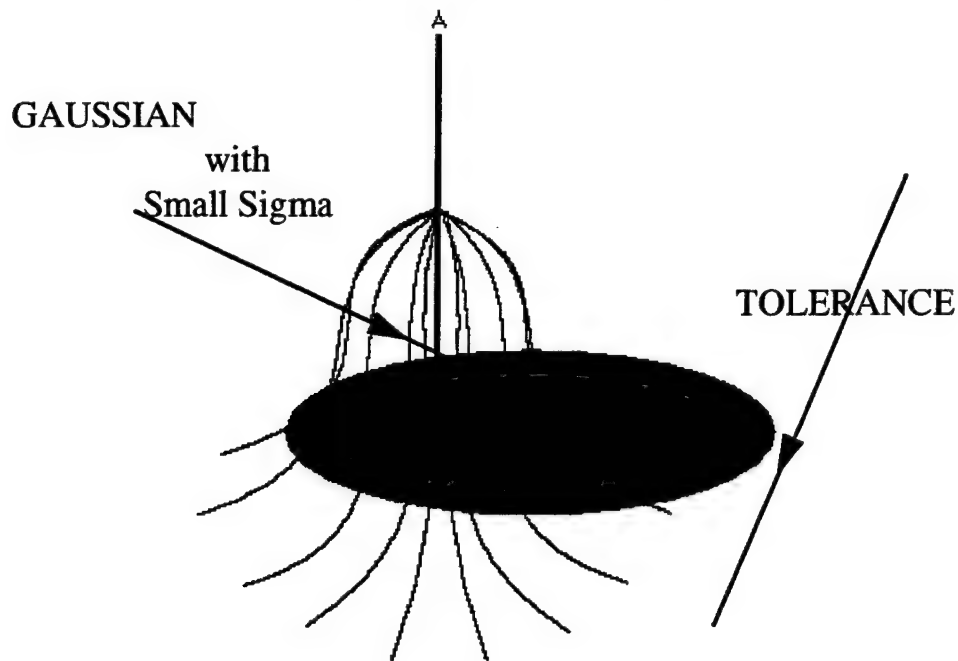


Figure 26

The shape and size of the error tolerance is determined by the vehicle type, elapsed time, and the accuracy of the sighting information. For example, if the gaussian circular-error-probable (CEP) of the sighting is known, the circular coverage function can be used to determine if the sighting is within a particular level of assurance of a circular tolerance shape. See Figure 27. In the process, the total probability distribution within the circle is integrated. If, say, a ninety percent assurance is required, then ninety percent of the distribution must lie within the circle. If so, no message is sent. The

automatic curve fitting mechanism then adjusts to the new position, recalculates the error shape, and continues. If not, the actual position is sent and the curve fitting adjusted.



A = actual sighting
P = predicted sighting

Figure 27, Assurance Calculations

The problem of predicting a location based on historical track information is non-trivial. Agent, for example, will have to know when a sighting is clearly improbable, and thus in error, so that it can be tacitly discarded. While for short intervals a linear prediction may be useful, for longer intervals a more complex prediction algorithm must be employed. One important study to be performed in the out years will be to determine the best prediction algorithms to employ. A fixed circle will be used as the error tolerance, along with a linear curve fit with zero uncertainty. However, this will by no means demonstrate the potential effectiveness of the extrapolation technique.

E. TRADITIONAL COMPRESSION

If the targeted processor can keep up with the data rate, a loss-less compression technique, such as Lempel-Ziv, can be applied to lengthy messages. The messages will be examined for good candidates for this type of compression, and estimates of the time necessary for the various algorithms to perform the compression will be estimated. When resources permit, an attempt to establish a benchmark for this type of compression will be made in the SIF.

With respect to video and voice, at this point it is believed that hardware assistance will be required for compression/decompression. This task is therefore limited to identifying all of the platform types that can be connected to the link, and determining, to the extent information is available, if COTS compression boards are available for these platforms, or will be available within a reasonable time span.

F. EXTENSIONS

The immediate application is to reduce the amount of redundant information transmitted over the link (see individual tasks below). However, additional autonomous agent functions could help to better manage the tactical links by detecting problems and taking steps to alleviate the consequent load on the net, when possible and acceptable.

The initial analysis and design of the intelligent agent architecture will also be directed towards meeting the system requirements for robustness in the face of communications gaps or failures. The architecture provides significant opportunity for extension. Out year tasks will develop agents that manage other types of messages, support recognition, routing and processing of new RTR message types, and call attention to messages requiring quick reaction.

Over the course of this work and subsequent efforts, all message types will be examined for ways in which a intelligent agent could optimize network operations.

This effort will include interviews of expert users of the system in an effort to collect additional cases or functions that are appropriate to intelligent agent action, and where appropriate seek to incorporate this in the current network schema.

VI. SUMMARY

A. WORK ACCOMPLISHMENTS

1. The engineering design of Agent has been documented to the point where detailed design can use it as a guide.
2. All of the software and functional requirements of Agent are satisfied by this design.
3. The Agent engineering design is not specific to Link-16 or any other tactical network.
4. Agent is compatible with a multi-processor implementation, but can run on a single-processor machine.
5. Agent has its own inference engine, the special features of which are described in detail.
6. Agent is otherwise compatible with IBM's Agent Building Environment (ABE). During detailed design, another tool kit can be substituted if found to be more appropriate.

B. WORK TO BE DONE IN SUBSEQUENT PHASES

1. Make sure that the rule bases are thoroughly debugged, especially with respect to NPG loading calculations. Construct a test bed in SNOBOL for this purpose.
2. Add rules for calculating node loading.
3. A low-level briefing on the engineering design should be written.
4. The low-level briefing should provide the basis for a conference to discuss the engineering design among the interested parties.
5. Add formal BNF to describe the facts and rules. Formalize the otherwise informal notations used in this document.
6. Create rules for passive net normalization.

7. Create rules for agent-to-agent "update me" messages for active normalization.
8. Insert sequence numbers into encoded messages (quasi token ring scheme) so that receiving nodes can detect missing updates.
9. Agent as written does not compare actual net loading with predicted net loading in order to do a kind of "sanity check" on its own operations. This can be added.
10. Encode and Decode rule bases for other heuristics need to be written.
11. A high-level briefing on the engineering design should be written.
12. A low-level briefing on the engineering design should be written.
13. A command-level briefing on Agent should be written.
14. At some point before detailed design, a survey of inference engines should be made to see if any are close enough to Agent requirements to be used as a point of departure.
15. At some point before detailed design, a survey of agent-building toolkits should be made.

VII. FUTURE WORK

Today's Link 16 is passive, for the most part requiring acknowledgments only for certain types of messages, such as military orders. However, with Agent in place, and Agent-to-Agent communications demonstrated, more complex ways of fine-tuning network operation are possible. In an example mentioned above, a platform could request complete historical information on a specific track, and receive that information in an efficient manner. Also, the frequency of updates could be controlled by the individual needs of the receiving platforms, with the platform requiring the most urgency receiving priority. Network optimization, while nevertheless working within today's rules and restraints, can be an operational reality.

In the out years, an attempt will be made to quantify the gain expected from allowing the Agents to communicate with each other, whereby they adjust network parameters in real time. For an example involving considerable intelligence, update rates could be dynamically adjusted by the situational assessment of the importance of a track relative to the tactical situation. This is a significant step towards more autonomous functionality based on an agent's own situation assessment.

Additional, more specialized agents may contain user-specific knowledge. For example, Agent could passively notify the special agent of events as they occur, and the special agent could provide various types of amplifying information to the console user based upon the operator's needs. The specialized agent may be capable of a range of responses, from simple alert messages to action sequences.

In the case of urgency-driven updates, the receiving platforms would examine the current rate at which a given track was being updated, and, if satisfactory, do nothing. If a higher rate of update is desirable, a Agent to Agent message is generated that increases the update rate. In this way, update rates could be drive up or down based upon the perceived tactical situation. A very intelligent and platform-specific Agent could do this function transparently.

Ideally, Agents would utilize a CORBA-compliant object-request broker (ORB), such as Iona's ORBIX, or possibly incorporate the JAVA applet to application

architecture, to transparently manage track updates (and, indeed, all message objects). Each C2P would, under an ORB, appear to have all track objects present in its own address space, regardless of where on the link they actually resided. Then, rather than transmit at fixed intervals, platforms could simply examine the track objects at whatever frequency is thought to be important. This would reverse the idea of the link—from fixed-interval transmission to as-needed query. The ORB would ensure that the data was current. Figure 7 shows a layered view of a component-based architecture.

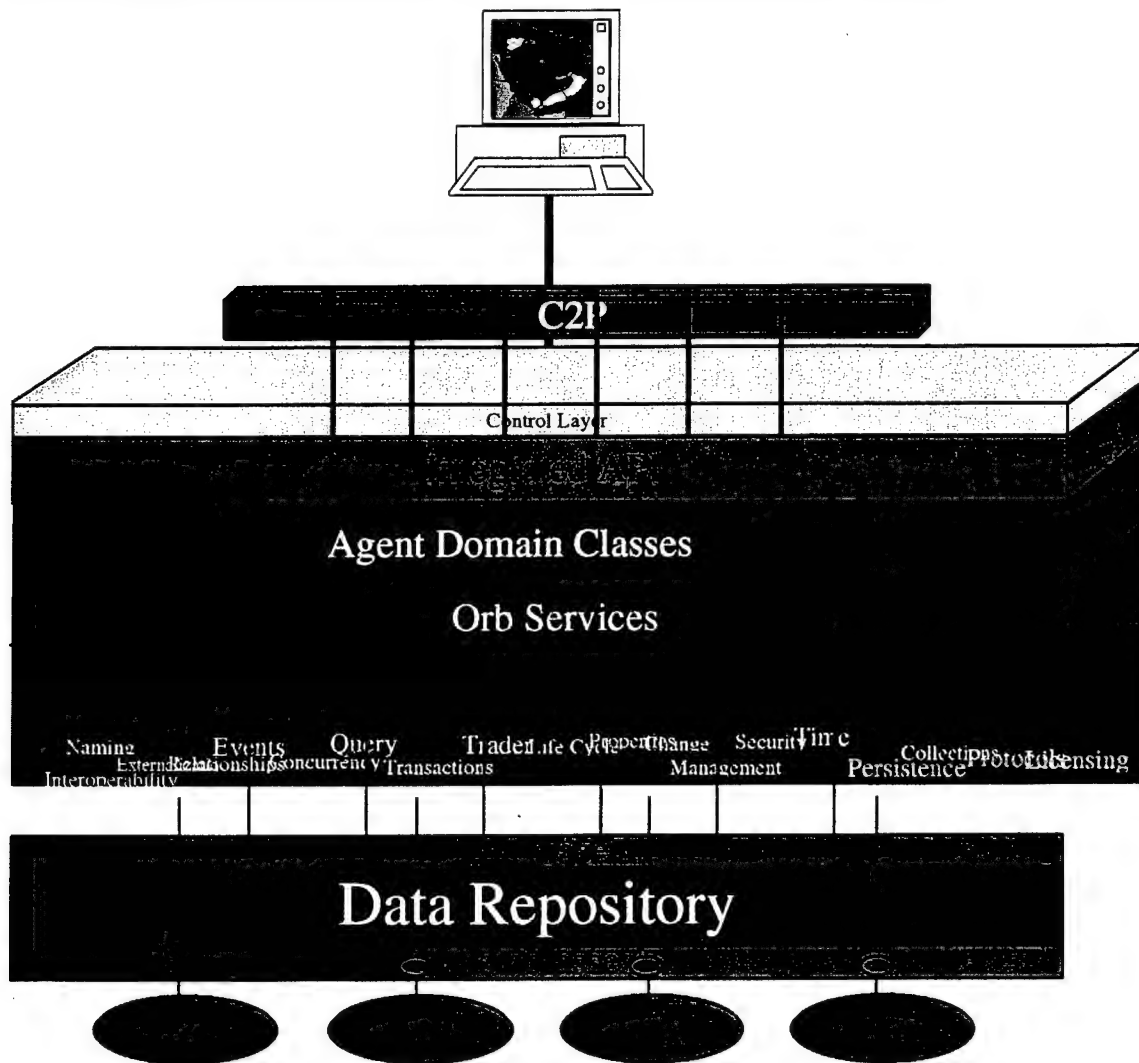


Figure 28, Orb-Based Architecture

As attractive as the use of an ORB-based system is, the current overhead of an ORB constitutes a significant chunk of bandwidth and may, with today's technology, slow down real-time processing, precluding its use in this context. The key to the analysis is at the system level, not at the link level.

Beyond the applicability to military communications, such technology has obvious implications with respect to commercial communications networks (INTERNET, or any other digital communications network). While it is acknowledged that the development of an agent architecture is not "the solution", it has the potential to alleviate already crowded communications lines, while not sacrificing accuracy. The robustness of such an approach has not yet been adequately determined. However, it is felt by the author that such an approach will provide for more optimized network utilization of available assets (bandwidth), ultimately resulting more efficient and overall faster network operations.

REFERENCES

- [1] Virdhagriswaran, Sankar, *Heterogeneous Information Systems Integration*. Crystaliz Inc. paper September 29, 1994
- [2] Foner, Leonard N, *Clustering and Information Sharing in an Ecology of Cooperating Agents* Agents Group, MIT Media Lab. E15-305, 20 Ames St., Cambridge, MA 02139. paper 1994-5.
- [3] Coriat, Michel, *Formal Specification Using Agents Conceptualization* Laboratoire MASI, Institut Blaise Pascal / CNRS-UA 818, Universite de PARIS VI, 4 place Jussieu 75252 Paris cedex 05, France. paper 1995.
- [4] Foner, Leonard N., *What's an Agent Anyway? A sociological Case Study*. Agents Group, MIT Media Lab, paper Agent Memo 93-01, 1993.
- [5] Beymer, David, *Pose-Invariant Face Recognition Using Real and Virtual Views*. A.I. Technical Report No. 1574, Massachusetts Institute of Technology, Artificial Intelligence Labatory, paper March 1996.
- [6] Maes, Pattie, *Modeling Adaptive Autonomous Agents*. MIT Media-Labatory, 20 Ames Street, Rm 305, Cambridge, MA. paper 1994.
- [7] Leitman, Robert, *Integrating HTTP with ATM*. Master's Thesis Mathematics, University of Waterloo, Ontario, Canada, paper 1995.
- [8] Coen, Michael H., *SodaBot: Agent Environment and Construction System*. MIT Artificial Intelligence Labatory, Cambridge, MA. paper September 14, 1994.
- [9] Tambe, Milind, et al. *Building Believable Agents for Simulation Environments: Extended Abstract*. Information Sciences Institute, University of Southern California and Artificial Intelligence Laboratory, University of Michigan, paper 1994.
- [10] Kotay, Keith D., et al. *Transportable Agents*. Department of Computer Science, Dartmouth College, paper November 10, 1994.

- [11] Gruber, Thomas R., *The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases*. Knowledge Systems Laboratory, Stanford University, paper January 31, 1991.
- [12] Hayes-Roth, Barbara, et al. *A Satisficing Cycle for Real-Time Reasoning in Intelligent Agents; Expert Systems with Applications*. Knowledge Systems Laboratory, Stanford University, paper 1993.
- [13] Finin, Tim, et al. *DRAFT Specification of the KQML Agent-Communication Language*. The DARPA Knowledge Sharing Initiative External Interfaces Working Group, paper June 15, 1993.
- [14] Moukas, Alexandros, *Amalthea: Information Discovery and Filtering using a Multiagent Evolving Ecosystem*. MIT Media Laboratory, Cambridge, MA, paper 1995.
- [15] Thirunavukkarasu, Chelliah, et al. *Secret Agents - A Security Architecture for the KQML Agent Communication Language*. Enterprise Integration Technologies, Computer Science and Electrical Engineering, University of Maryland, paper December 1995.
- [16] Genesereth, Michael R, et al. *A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation*. Computer Science Department, Stanford University, paper November 15, 1994.
- [17] Maes, Pattie, et al. *Kasbah: An Agent Marketplace for Buying and Selling Goods*. MIT Media Lab, paper 1996.
- [18] Tambe, Milind, et al. *Constraints and Design Choices in Building Intelligent Pilots for Simulated Aircraft: Extended Aircraft*. Information Systems Institute, University of Southern California, paper 1996.
- [19] Mayfield, James, et al. *Desiderata for Agent Communication Languages*. Computer Science Department, University of Maryland, paper 1995.
- [20] Finin, Tim, et al. *A Language and Protocol to Support Intelligent Agent Interoperability*. University of Maryland, paper April 1992.
- [21] Foner, Leonard N., *Clustering and Information Sharing in an Ecology of Cooperating Agents*. Agents Group, MIT Media Lab, paper 1995.

- [22] Bocionek, Siegfried, et al. *Dialog-Based Learning (DBL) for Adaptive Interface Agents and Programming-by-Demonstration Systems*. School of Computer Science, Carnegie Mellon University, paper July 1993.
- [23] McKay, Donald P, et al. *An Architecture for Information Agents*. Loral Defense Systems and Computer Sciences and Electrical Engineering, University of Maryland, paper 1995.
- [24] Pitt, Jeremy, et al. *Autonomous Agents in Inter-Organization Project Management*. Department of Computing, Imperial College of Science, Technology, and Medicine, UK, paper 1995.
- [25] Beymer, David James, *Pose-Invariant Face Recognition Using Real and Virtual Views*. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Dissertation September 1995.
- [26] Kunz, Thomas, *Abstract Behavior of Distributed Executions with Applications to Visualization*. Vom Fachbereich Informatik, der Technischen Hochschule Darmstadt, Dissertation May 6 1994.
- [27] Voorhees, Ellen M., *Agent Collaboration as a Resource Discovery Technique*. Siemens Corporate Research Inc., paper 1993.
- [28] Ooi Joo Li, *Access Control for an Object-Oriented Distribution Platform*. 1993. Thesis submitted University of Dublin, Trinity College for the degree of Master of Science (Computer Science)
- [29] Brunsting Raymond J., *Quality of Service Issues in Wireless Networks*. 1995. Thesis submitted University of Waterloo, Ontario, Canada, for the degree of Master of Mathematics (Computer Science)
- [30] Beale Andrew and Wood Andrew, *Agent Based Interaction*. People and Computers IX. Proceedings of HCP94, Glasgow, UK, pp239-245
- [31] Maes, Pattie, *Intelligent Software; Programs that can act independently will ease the burdens that computers put on people*. paper 1995. Massachusetts Institute of Technology

- [32] Franklin Stan and Graesser Art, *Is it an Agent, or just a Program? A taxonomy for Autonomous Agents*. 1996. Proceedings of the Third International Workshop on Agent Theories Architectures, and Languages, Springer-Verlag, 1996
- [33] Maes Pattie, *Artificial Life Entertainment: Lifelike Autonomous Agents*. paper 1996. Massachusetts Institute of Technology
- [34] Hayes-Roth Barbara, et. al. *A Domain-Specific Software Architecture for Adaptive Intelligent Systems*. April 1995. IEEE Transactions on Software Engineering, Vol. 21, No. 4, pp288-301
- [35] Featherstone Roy, *Robot Dynamics Algorithms*. Kluwer Academic Publishers 1987.
- [36] Valavanis Kimon P. and Saridis George N., *Intelligent Robotic Systems: Theory, Design, and Applications*. Kluwer Academic Publishers 1992.
- [37] DaBose, Michael W., *Tactical Link Object Oriented Technology Insertion*. paper 1995. Naval Research Development Test and Evaluation NRaD
- [38] DaBose, Michael W., *Tactical Link Object Oriented Technology Insertion*, paper 1996.
- [39] Chavez, Anthony, et al. *Challenger: A Multi-agent System for Distributed Resource Allocation*. Autonomous Agents Group, MIT Lab, paper 1996.
- [40] Vreeswijk, Gerard A. W., *Self-government in multi-agent systems: experiments and thought experiments*. University of Limburg, Department of Computer Science (FdAW), The Netherlands, paper April 7, 1995.
- [41] Vreeswijk, Gerard A. W., *Open Protocol in Multi-agent Systems*. University of Limburg, Department of Computer Science (FdAW), The Netherlands, paper January 1995.
- [42] Genesereth, Michael R. and Fikes, Richard E., et al. *Knowledge Interchange Format Version 3.0 Reference Manual*. Computer Science Department, Stanford University, paper June 1992.
- [43] Beale, Russell and Wood, Andrew, *Agent-Based Interaction* , in People and Computers IX: Proceedings of HCI'94, Glasgow, UK, pp. 239-245.

- [44] Durham, Jayson and Torrez, William, *A Neuron Model Using Cumulative Distribution Functions*, Proc. Of The World Congress on Neural Networks, Portland, OR.
- [45] Durham, Jayson, Gillcrist, Brenda, and Heckman, Paul, *A Testbed Processor for Embedded Multi-Computing*, Proc. Of The 6th International Symposium on Unmanned Untethered Submersible Technology, Washington, DC.
- [46] Durham, Jayson, *Engineering Intelligent Undersea Vehicles*, Proc. of OCEANS'89, September 18-21.
- [47] Durham, Jayson, Heckman, Paul, Bryan, Dale, and Riech, Ron, *EAVE-West: A Testbed for Plan Execution*, Proc. Of 5th International Symposium on Unmanned Untethered Submersible Technology, Univ. of New Hampshire, Durham, NH.
- [48] Franklin, Stan and Graesser, Art, *Is it and Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proc. of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag.
- [49] Hamming, Richard W., *Coding and Information Theory*, Prentice Hall, 1996
- [50] Krichevsky, Rafail, *Universal Compression and Retrieval*, Kluwer Academic Publishers, 1994
- [51] Kapur, J. N., *Measures of Information and Their Applications*, John Wiley & Sons, 1994
- [52] Kapur, J. N., *Maximum-Entropy Models in Science and Engineering*, John Wiley & Sons, 1989
- [53] Maes, Pattie, *Intelligent Software*, Scientific American, 1995
- [54] Maes, Pattie, *Artificial Life Meets Entertainment: Lifelike Autonomous Agents*, CACM, 1995
- [55] Nelson, Mark, and Gailly, Jean-Loup, *The Data Compression Book, 2nd Edition*, M&T Books, 1996
- [56] Shannon, Claude, *Mathematical Theory of Communication*, 1948
- [57] Tenenbaum, Andrew S., *Computer Networks, 3rd Edition*, Prentice Hall, 1996

- [58] Torres, William and Durham, Jayson, *On Fitting Transformed Distributions to Empirical Data*, INTERFACE 1993.
- [59] Meystel, A., *Autonomous Mobil Robots*, World Scientific, 1991
- [60] Valavanis, Kimon P., Saridis, George N., *Intelligent Robotic Systems: Theory, Design, and Applications*, Kluwer Academic Publishers, 1992
- [61] de Silva, Clarence W., *Intelligent Control Fuzzy Logic Applications*, CRC Press Inc., 1995
- [62] Lima, P U, Saridis, G N., *Design of Intelligent Control Systems Based on Hierarchical Stochastic Automata*, Rensselaer Poly Institute, 1996
- [63] Huang, Hui-Min, *An Operator Experience with a Heirarchical Real-Time Control System (RCS)*, National Institute of Standards and Technology, 1996

APPENDIX

CONCEPTUAL PROTOTYPE SOURCE CODE

Note - This prototype was developed utilizing:

- Intel Pentium 166Mhz processor
- Windows 95 OS
- Rogue Software gui builder and TCP/IP sockets development packages
- Borland C++ version 4.51 compiler

Copyright material (header files included with the above build packages) are omitted.

```
typedef void (* ENTRY)(void *);
```

```
class CService{
```

```
private:
```

```
LPSTR  
HANDLE
```

```
SERVICE_STATUS_HANDLE
```

```
BOOL  
BOOL  
HANDLE
```

```
m_ServiceName;
```

```
m_ExitEvent;
```

```
m_ServiceStatusHandle;
```

```
m_PauseService;
```

```
m_RunningService;
```

```
m_Threathandle;
```

```

ENTRY                                     m_EntryFunction;

int                                     m_StartTimeout;
int                                     m_StopTimeout;
int                                     m_PauseTimeout;
int                                     m_ResumeTimeout;

void OnPauseService();
void OnResumeService();
void OnStopService();
long OnStartService();

static VOID ServiceMain(DWORD argc, LPTSTR *argv);

BOOL SCMStatus (DWORD dwCurrentState,DWORD dwWin32ExitCode,
                DWORD dwServiceSpecificExitCode,DWORD dwCheckPoint,DWORD dwWaitHint);

static VOID ServiceCtrlHandler (DWORD controlCode);

VOID Exit(DWORD error);

//install variables
DWORD m_dwDesiredAccess;
DWORD m_dwServiceType;
DWORD m_dwStartType;
DWORD m_dwErrorControl;

LPSTR m_szLoadOrderGroup;
LPDWORD m_lpdwTagID;
LPSTR m_szDependencies;

public:
CService();
~CService();

//service entry point
long InitService(LPCSTR name,void * EntryFunction);

//service timeout settings
int SetStartTimeOut(int milisec);
int SetStopTimeOut(int milisec);
int SetPauseTimeOut(int milisec);
int SetResumeTimeOut(int milisec);

//service install / un-install
int SetInstallOptions(DWORD dwDesiredAccess, DWORD dwServiceType,
                     DWORD dwStartType,DWORD dwErrorControl);
int SetInstallOptions(LPSTR szLoadOrderGroup,LPDWORD lpdwTagID,
                     LPSTR szDependencies);
int InstallService(LPCSTR szInternName,LPCSTR szDisplayName,LPCSTR szFullPath);
int InstallService(LPCSTR szInternName,LPCSTR szDisplayName,LPCSTR szFullPath,
                  LPCSTR szAccountName,LPCSTR szPassword);
int RemoveService(LPCSTR szInternName);

};

#include "zapp.hpp"
#include "mdiapp.hpp"                                     // Main App Class Definitions & Includes

ZAPP_IMPL_ASSERTS

// zpb_begin GlobalVars

```

```

    HWND status_hWnd;
    char stuff[LBUFFSZ];

    char *DxBuf[DXBUFSZ];
    int DxBufInp;
    int DxBufNdx;

//
// SlpQ is an object which allows independent execution of member
// functions.
//

class SlpQ {
    #define NTASKS 50
    unsigned long SlTm[NTASKS];
    unsigned long WkTm[NTASKS];
    zMDIChildFrame *sq[NTASKS];

public:
    SlpQ();
    ~SlpQ();
    unsigned long systime;
    int Slp(zMDIChildFrame *, unsigned long);
    int CkQ();
    int Rm(zMDIChildFrame *);
}; //end SlpQ

DWORD WINAPI ThreadFunc(LPVOID lpvThreadParm); //forward reference
SlpQ *pSQ; //pointer to the sleep queue

// end sleep queue parameters

// *****
// Track Class Declaration
// *****
#define MXTKMS 10000
#define MXTKPTS 24
#define HOSTILE_AIR 0
#define HOSTILE_SUB 1
#define HOSTILE_SURFACE 2
#define UNKNOWN_AIR 3
#define UNKNOWN_SUB 4
#define UNKNOWN_SURFACE 5
#define FRIENDLY_AIR 6
#define FRIENDLY_SUB 7
#define FRIENDLY_SURFACE 8

class Trk {
public:
    Trk();
    ~Trk();
    int upd(); //update the track info for display purposes
    int dmsg(); //delta message method
    int type; //type of track
    int num; //track number
    int xpos; //current x location for display on track window
    int ypos; // " y " " " " " "
    int xvec; // x location of track vector endpoint
    int yvec; // y " " " " "
    float altRaw; // raw altitude value directly converted
    float xRaw; // raw coordinate value
    float yRaw; // raw coordinate value
    float spRaw; // raw speed value directly converted
    float xdRaw; // raw delta of coordinate value
    float ydRaw; // raw delta of coordinate value
    zColor clr; //color of display is initially white
    char timStr[16]; //timestamp is the string directly from the msg
    char typStr[8]; //type of track
    char strStr[32]; //undefined characters

```

```

char numStr[8];          //track number
char romStr[LBUFSZ];     //rest of message string
char OldMsg[LBUFSZ];     //Old Message as copied directly out of buffer
char CurMsg[LBUFSZ];     //New " " " " " " "
int OldDel[LBUFSZ];      //Changed symbols for old message
int CurDel[LBUFSZ];      //Changed symbols for current message
int MQndx;               //index to the latest entry in MQ
char *MQ[MXTKMS];        //message buffer for this track
int cCnt[MXTKMS];        //symbol count per message, i.e. length
int dCnt[MXTKMS];        //diff value count between messages
float csRatio[MXTKMS];   //ratio of diff/total count
WDataAn *AnWn;           //Analysis window for Track Txt window
WTrkTxt *TxWn;           //this is a text window for the message buffer
int distype;             //this is a unknown of display bitmap
char dTypStr[64];        //string for the type of displayed icon
}; //end Trk

// *****
// Track Queue Class Declaration
// *****
#define MAXTRKS 400                      //Maximum number of tracks per Q
class TrkQ {
public:
    Trk *TQ[MAXTRKS];                  //Array of Pointers to Tracks
    Trk *newTrk;                        //reserve track
    int nTrks;                          //number of tracks in the queue
    int DTindx;                         //index for the track information dialog
    int newTrkTxt;                      //index for text message display for tracks
    int xdTrkTxt;                      //x offset for new TrkTxt window
    int ydTrkTxt;                      //y offset for new TrkTxt window
    int newTrkAn;                      //index for the track analysis window
    TrkQ();
    ~TrkQ();
    int upd(char *);                   //this is for display coordinates
    float minlg;
    float maxlg;
    float dellg;
    float minlt;
    float maxlt;
    float dellt;
    int rmv(zString *);               //removes the track in the message string
    int rescale();                   //resets the display coordinates
    //Trk *ptrk;                      //global (common) ptr to a current DTInfo Trk
}; //end TrkQ

// *****
// Code Book Class Declaration
// *****
#define MAXFMTS 32
#define MAXFLDS 32
class CdBk {
public:
    CdBk();
    ~CdBk();
    int ffmt[MAXFMTS];
    zString *flds[MAXFLDS];
}; //end CdBk Class Declaration

// *****
// Track Methods
// *****
Trk::Trk(){
    type = 0;                        //type of track
    num = 0;                         //track number

```

```

xpos = 0;           //current x location for display on track win
ypos = 0;           // " y " " " " " " "
clr = WHITE;        //color of display is initially white
distype = UNKNOWN_SURFACE; //initialized of bitmap is unknowtype
strcpy(dTypStr,"Unknown Surface"); //string for the type of icon displayed
strcpy( timStr,""); //timestamp is the string directly from the msg
strcpy( typStr,""); //type of track
strcpy( strStr,""); //undefined characters
strcpy( numStr,""); //track number
strcpy( romStr,""); //rest of message string
strcpy( OldMsg,""); //Old Message as copied directly out of buffer
strcpy( CurMsg,""); //New " " " " " " " "
MQndx = 0;          //index to the latest entry in MQ
int ndx;             //tmp index for for-loop
for(ndx=0;ndx<MXTKMS;ndx++){
    MQ[ndx] = NULL; //message buffer for this track
} //endfor
TxWn = NULL;        //the text display window is initially NULL
} //end Trk constructor

Trk::~Trk() {
} //end Trk destructor

int Trk::upd() { //update the particular track
    return 0;
} //end Trk update

int Trk::dmsg() {
    //local constants
#define N_TKMSGs 24 // total number of messages

    //local variables
    int tmp_ndx; //temporary index
    int lbufNdx;
    //int r_curr = MQndx; //actual index into the buffer of messages

    char *ptc1;
    char *ptc0;
    char *ptcm1;
    int numchrs;
    int ndx;
    int dsum;
    int scnt[LBUFSZ];
    char netbuf[LBUFSZ];

    if(MQndx>2){ //we need three messages for delta messaging
        ptc1 = MQ[MQndx-1]; //get pointer to message
        ptc0 = MQ[MQndx-2];
        numchrs = strlen(ptc1);
        dsum = 0;
        int ccnt;
        for(ccnt=0;(ccnt<numchrs)&&(ccnt<LBUFSZ);ccnt++){
            if((*ptc1) != (*ptc0)){
                scnt[ccnt] = 1;
                CurDel[ccnt] = 1;
                dsum++; //this counts number of chars not same
            } else {
                scnt[ccnt] = 0;
                CurDel[ccnt] = 0;
            } //endif
            ptc1++;
            ptc0++;
        } //endfor

        ptc0 = MQ[MQndx-2]; //reset pointer to line buffers
    }

```



```

ptcm1 = MQ[MQndx-3];
for(ccnt=0;(ccnt<numchrs)&&(ccnt<LBUFSZ);ccnt++){
    if((*ptc0) != (*ptcm1)){
        OldDel[ccnt] = 1;
    }else{
        OldDel[ccnt] = 0;
    }//endif
    ptc0++;
    ptcm1++;
}

//following section of code is an initial preliminary coding
//for run length encoded "delta messages"
//note that the major problem for this is that the overhead
//greatly reduces the number of symbols saved.
//run-length encoding is currently turned off.
#define MXCHGS 10
#define MXRN 16
int dmsg[MXCHGS]; // = {0,0,0,0,0,0,0,0,0,0};
int dmsgndx = 0; // test to see if we have a delta message
int tmdel = strlen(timStr) + 1;
int oldccnt;
for(ccnt=tmdel;
    ((ccnt<numchrs)&&(ccnt<LBUFSZ));
    ccnt++){
    if ( CurDel[ccnt] == 1 ) //we have a new char
        &&( OldDel[ccnt] == 0 ) { //no delta message
        oldccnt = dmsg[dmsgndx];
        dmsg[dmsgndx] = ccnt;
        if(dmsgndx<MXCHGS){
            dmsgndx++;
        }//endif
    }//endif
}

ptc1 = MQ[MQndx-1]; //reset pointer to curr buffer
ptc1 += tmdel;
if(dmsgndx <= 0){ //we have a delta message
    int ntbfindx = 0;
    int tndx;
    for(tndx=0;tndx<dmsgndx;tndx++){
        netbuf[ntbfindx] = (*(ptc1+dmsg[tndx]));
    }//endif
    ntbfindx++;

    netbuf[ntbfindx] = '!';
    ntbfindx++;

    for(ccnt=tmdel;
        (ccnt<numchrs)&&(ccnt<LBUFSZ);
        ccnt++){
        if( OldDel[ccnt] == 1 ){ //delta char
            netbuf[ntbfindx] = *ptc1;
            ntbfindx++;
        }//endif
        ptc1++;
    }//endif
    netbuf[ntbfindx] = '\0';

    //the following for loops are here because extraneous characters
    //were in the resultant string if the strxxx() routines are used
    //also, if strcpy() is used instead of strncpy(), very strange
    //looking oversized strings were created. ??? library or compiler ???
    char dmbuf[LBUFSZ] = " ";
    //strcpy(dmbuf,timStr);
    int tmpndx;
    for(tmpndx=0;tmpndx<12;tmpndx++){

```

```

        (*(dmbuf+tmpndx)) = (*(timStr+tmpndx));
    }//endfor
    //strcat(dmbuf,numStr);
    for(tmpndx=0;tmpndx<3;tmpndx++){
        (*(dmbuf+12+tmpndx)) = (*(numStr+tmpndx));
    }//endfor
    //strcat(dmbuf,netbuf);
    for(tmpndx=0;tmpndx<=ntbndx;tmpndx++){
        (*(dmbuf+15+tmpndx)) = (*(netbuf+tmpndx));
    }//endfor

    //the following sends a track update message to the other machine
    C_WSClient client;    //create the object RTRclient
    if (client.Connect(79,"128.49.133.12") != WSC_SUCCESS)
        MessageBox(NULL,"Connection Fail"," ",MB_OK);
    //endif
    client.Send(dmbuf);
    client.Send("\r\n");
    client.CloseConnection();

```

```

} else { //send raw message

```

```

    //the following sends a track update message to the other machine
    C_WSClient client;    //create the object RTRclient
    if (client.Connect(79,"128.49.133.12") != WSC_SUCCESS)
        MessageBox(NULL,"Connection Fail"," ",MB_OK);
    //endif
    //strcpy(netbuf, MQ[MQndx-1],90);
    client.Send(MQ[MQndx-1]); //netbuf);
    client.Send("\r\n");
    client.CloseConnection();

```

```

} //endif

```

```

} else { //first two messages pass through

```

```

    //the following sends a track update message to the other machine
    C_WSClient client;    //create the object RTRclient
    if (client.Connect(79,"128.49.133.12") != WSC_SUCCESS)
        MessageBox(NULL,"Connection Fail"," ",MB_OK);
    //endif
    //strcpy(netbuf, MQ[MQndx-1],90);
    client.Send(MQ[MQndx-1]); //netbuf);
    client.Send("\r\n");
    client.CloseConnection();

```

```

} //endif

```

```

return 1;

```

```

}

```

```

TrkQ::TrkQ(){
    int ndx;
    for(ndx=0;ndx<MAXTRKS;ndx++){ //initialize the track array
        TQ[ndx]=NULL;
    }//endfor
    newTrk = new Trk;
    nTrks = 0;
    minlt = 19.0;
    maxlt = 23.0;
    minlg = -162.0; //this is for op area field of view
    maxlg = -158.0;
    dellt = maxlt - minlt;

```

```

    dellg = maxlg - minlg;
    newTrkTxt = 0; //default for null number of TrkTxt windows
    xdTrkTxt = 0; //initialize to no delta for TrkTxt windows
    ydTrkTxt = 0;
} //end TrkQ constructor

TrkQ::~TrkQ() {
    int ndx;
    for(ndx=0; ndx<MAXTRKS; ndx++){ //delete the track array
        if(TQ[ndx] != NULL) {delete TQ[ndx];}
    } //endfor
    nTrks = 0;
} //end TrkQ destructor

int TrkQ::upd(char *pTstr) {
    if( *(pTstr + 16) == '2') { //verify valid message type
        strncpy(newTrk->typStr, pTstr + 15, 4); //type of track
        strncpy(newTrk->numStr, pTstr + 40, 3); //track number
        int tmpnum;
        tmpnum = atoi(newTrk->numStr);
        int ndx = 0;
        while( (ndx < MAXTRKS) //search for existing track
            &&(TQ[ndx] != NULL) ){
            if(TQ[ndx]->num == tmpnum){break;} //endif this track exists
            ndx++;
            //note problem with testing NULL ptr
        } //endwhile

        if(TQ[ndx] == NULL) {
            TQ[ndx] = newTrk; //we are adding a new track
            TQ[ndx]->num = tmpnum;
            nTrks++; //need to add error handling for case of MAXTRKS
            newTrk = new Trk;
        } else {
            strcpy(TQ[ndx]->OldMsg, TQ[ndx]->CurMsg);
        } //endif

        strncpy(TQ[ndx]->timStr, pTstr + 12, 12); //timestamp of track message
        //strcat(TQ[ndx]->timStr, "0"); //end-of-string
        strncpy(TQ[ndx]->typStr, pTstr + 15, 4); //type of track
        strncpy(TQ[ndx]->strStr, pTstr + 21, 8); //undefined characters
        strncpy(TQ[ndx]->numStr, pTstr + 40, 3); //track number
        strcpy(TQ[ndx]->romStr, pTstr + 43); //rest of message string
        char tmpstr[64];
        if(strcmp(TQ[ndx]->typStr, "02.2") == 0) {
            strncpy(tmpstr, pTstr + 50, 10);
            TQ[ndx]->altRaw = atof(tmpstr);
        } else {
            TQ[ndx]->altRaw = 0.0;
        } //endif
        float tmpx = TQ[ndx]->xRaw;
        float tmpy = TQ[ndx]->yRaw;
        strncpy(tmpstr, pTstr + 62, 7);
        TQ[ndx]->xRaw = atof(tmpstr);
        TQ[ndx]->xdRaw = TQ[ndx]->xRaw - tmpx;
        strncpy(tmpstr, pTstr + 70, 9);
        TQ[ndx]->yRaw = atof(tmpstr);
        TQ[ndx]->ydRaw = TQ[ndx]->yRaw - tmpy;
        strncpy(tmpstr, pTstr + 80, 9);
        TQ[ndx]->spRaw = atof(tmpstr);
        strcpy(TQ[ndx]->CurMsg, pTstr);
        if(TQ[ndx]->MQndx < MXTKMS) {
            TQ[ndx]->MQ[TQ[ndx]->MQndx] = pTstr;
            TQ[ndx]->MQndx++;
        } //endif

        TQ[ndx]->dmsg(); //send the message via delta messaging
    }
}

```

```

    }
    else
    {
        //the following sends a track update message to the other machine
        C_WSCClient client; //create the object RTRclient
        if (client.Connect(79,"128.49.133.12") != WSC_SUCCESS)
            MessageBox(NULL,"Connection Fail"," ",MB_OK);
        //endif
        client.Send(pTstr);
        client.Send("\r\n");
        client.CloseConnection();

    } //endif
    return 0;
} //end TrkQ update

int TrkQ::rescale()
{
    if( nTrks > 1){

        minlt=TQ[0]->xRaw;
        maxlt=TQ[0]->xRaw;
        minlg=TQ[0]->yRaw;
        maxlg=TQ[0]->yRaw;
        int ndx00;
        for(ndx00=0;ndx00<nTrks;ndx00++){
            if(minlt > TQ[ndx00]->xRaw){minlt=TQ[ndx00]->xRaw;} //endif
            if(maxlt < TQ[ndx00]->xRaw){maxlt=TQ[ndx00]->xRaw;} //endif
            if(minlg > TQ[ndx00]->yRaw){minlg=TQ[ndx00]->yRaw;} //endif
            if(maxlg < TQ[ndx00]->yRaw){maxlg=TQ[ndx00]->yRaw;} //endif
        } //endfor
        dellt = maxlt - minlt;
        dellg = maxlg - minlg;
        #define SCFAC 0.5
        if(dellt > 0.0){
            minlt -= (SCFAC * dellt);
            maxlt += (SCFAC * dellt);
        } else {
            minlt = (SCFAC * minlt);
            maxlt += (SCFAC * maxlt);
        } //endif
        if(dellg > 0.0){
            minlg -= (SCFAC * dellg);
            maxlg += (SCFAC * dellg);
        } else {
            minlg = (SCFAC * minlg);
            maxlg += (SCFAC * maxlg);
        } //endif
        dellt = maxlt - minlt;
        dellg = maxlg - minlg;

    } //endif

    return 0;
} //end rescale()

TrkQ *pTQ; //pointer to the track queue

// zpb_end

// ToolBar Member Functions - MainTools
tbMainMainTools::tbMainMainTools(zMDIMarginFrame *w, zSizer *sz, zBitmap *bmp)
: zToolBar(w, sz) {
    theBmp = bmp;
    (ZNEW zToolButton(this, ZNEW zSizer(zPoint(11,2), zDimension(24,22)),
    0, IDM_HELPCONTENTS, theBmp, zRect(112, 0, 128, 15)))
    ->show();
    (ZNEW zToolButton(this, ZNEW zSizer(zPoint(48,2), zDimension(24,22)),
    0, IDM_RTRDXTEXT, theBmp, zRect(0, 0, 16, 15)))

```

```

        ->show();
        // zpb_begin tbMainMainToolsConstructor
        // zpb_end
    }

tbMainMainTools::~tbMainMainTools() {
    // zpb_begin tbMainMainToolsDestructor
    // zpb_end
    if (theBmp)
        delete theBmp;
}

//
// Frame Member Functions - Main
//
// Window Constructor
WMain::WMain(const char *title)
: zMDIAppFrame(0,ZNEW zSizer(),zSTDFRAME,
    title, ZNEW zMenu(zResId(IDM_MDIWindowMenu))) {
    int MDIWindowPos = 1;
    // zpb_begin WMainConstructor2
    // zpb_end
    menu(ZNEW zMenu(this, zResId(IDM_MainMnMenu)));
    if (MDIWindowPos >= 0)
        menu()->insertDropDown(MDIMenu(), zString(zResId(IDS_MDIWINDOWMENU)), MDIWindowPos);
    else
        menu()->appendDropDown(MDIMenu(), zString(zResId(IDS_MDIWINDOWMENU)));
    menu()->setCommand(this, (CommandProc)&WMain::cmdMDIWindow, IDM_CASCADE, IDM_ARRANGEICONS);
    menu()->setCommand(this, (CommandProc)&WMain::cmdRTRDxDel, IDM_RTRDXDEL);
    menu()->setCommand(this, (CommandProc)&WMain::cmdRTRDxText, IDM_RTRDXTEXT);
    menu()->setCommand(this, (CommandProc)&WMain::cmdRTRDxSym, IDM_RTRDXSYM);
    menu()->setCommand(this, (CommandProc)&WMain::cmdRTRDxTrck, IDM_RTRDXTRCK);
    menu()->setCommand(this, (CommandProc)&WMain::cmdHelpContents, IDM_HELPCONTENTS);
    menu()->setCommand(this, (CommandProc)&WMain::cmdHelpAbout, IDM_HELPABOUT);
    // Create Tool Bar Margin Frame
    zMDIMarginFrame *ToolFrame = ZNEW zMDIMarginFrame(this,
        ZNEW zGrowToFitSizer(ZGRAV_TOP, sizer()));
    ToolFrame->show();
    pTB = ZNEW tbMainMainTools(ToolFrame, ZNEW zGravSizer(ZGRAV_TOP, 30, ToolFrame->sizer()), ZNEW
        zBitmap(zResId(IDB_MainMainTools)));
    pTB->show();
    zMDIMarginFrame *sf=ZNEW zMDIMarginFrame(this,ZNEW zGrowToFitSizer(ZGRAV_BOTTOM,sizer()));
    zStatusLineEZ* pSB = ZNEW zStatusLineEZ(sf, ZNEW zGravSizer(ZGRAV_BOTTOM,0, sf->sizer()), ZSL_STANDARDITEM);
    sf->show();
    pSB->show();
    ZNEW Wjtidssl((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_JTIDSSL)));
    // zpb_begin WMainConstructor1
    // zpb_end
    // zpb_begin WMainConstructor

    pSQ= new SipQ; //creates the sleep queue for periodic execution of
                    //member functions.

    pTQ = new TrkQ; //creates the track queue for this window

    int x = 0;
    DWORD dwResult = 0;
    DWORD dwThreadId;
    HANDLE hThread;

    hThread = CreateThread(NULL, 0, ThreadFunc, (LPVOID) &x,
        0, &dwThreadId);
    SetThreadPriority(hThread,THREAD_PRIORITY_ABOVE_NORMAL);

    // zpb_end
    show(SW_MAXIMIZE);

```

```

}

WMain::~WMain() {
    // zpb_begin WMainDestructor1
    // zpb_end
}

//
// Menu Item Selection Handlers
//
int WMain::cmdMDIWindow(zCommandEvt* ev) {
    // zpb_begin WMainMDICmds
    switch (ev->cmd()) {
        case IDM_CASCADE:
            cascade();
            break;
        case IDM_TILE:
            tile();
            break;
        case IDM_ARRANGEICONS:
            arrangeIcons();
            break;
    }
    // zpb_end
    return 0;
}

int WMain::cmdRTRDxDel(zCommandEvt* ev) {
    ZNEW WDelWin((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_DELWIN)));
    // zpb_begin WMainRTRDxDel
    // zpb_end
    return 0;
}

int WMain::cmdRTRDxText(zCommandEvt* ev) {
    ZNEW WDXWin((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_DXWIN)));
    // zpb_begin WMainRTRDxText
    // zpb_end
    return 0;
}

int WMain::cmdRTRDxSym(zCommandEvt* ev) {
    ZNEW WSymWin((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_SYMWIN)));
    // zpb_begin WMainRTRDxSym
    // zpb_end
    return 0;
}

int WMain::cmdRTRDxTrck(zCommandEvt* ev) {
    ZNEW WTrkWin((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_TRKWIN)));
    // zpb_begin WMainRTRDxTrck
    // zpb_end
    return 0;
}

int WMain::cmdHelpContents(zCommandEvt* ev) {
    // zpb_begin WMaincmdHelpContents
    // zpb_end
    return 0;
}

int WMain::cmdHelpAbout(zCommandEvt* ev) {
    DAbout* p=ZNEW DAbout(this, zResId(IDD_About));
    p->modal();
    if (p->completed()) {
        // zpb_begin WMainHelpAbout
        // zpb_end
    }
}

```

```

    } else {
        // zpb_begin WMainHelpAboutCancel
        // zpb_end
    }
    delete p;
    return 0;
}

// zpb_begin WMainMemberFunctions
// zpb_end

// Pane Member Functions - DxWinDxTxtPane
DxWinDxTxtPane::DxWinDxTxtPane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD) {

    // zpb_begin pDxWinDxTxtPaneConstructor1
    // zpb_end
    show();
}

int DxWinDxTxtPane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin DxWinDxTxtPaneSize
    // zpb_end
    return zWindow::size(ev);
}

int DxWinDxTxtPane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin DxTxtPaneDraw

    //local constants
#define N_DXMSGGS 18                // total number of messages displayed on the pane
#define x_head 3                    // x coordinate for the header row
#define y_head 10                   // y coordinate for the header row
#define x_mbase 3                   // x coordinate for base (first row) of display messages
#define y_mbase 10                  // y coordinate for base (first row) of display messages
#define m_strt 0                    // starting message number
#define r_offset 15                 //offset for spacing between rows

    //local variables
    int tmp_ndx;                    //temporary index
    int lbufNdx;
    int r_ndx;                      //row index for looping through the active messages currently displayed
    int r_curr;                     //actual index into the buffer of messages

    char *msg_curr[N_DXMSGGS];

    //set up the display parameters
    canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));

    //clear the window pane
    zRect DxTxtArea;
    canvas()->setVisible(DxTxtArea);
    //canvas()->pushPen(new zPen(zColor(GREEN), Solid, 5));
    canvas()->pushBrush(new zBrush(zColor(255,128,128)));
    canvas()->rectangle(DxTxtArea);
    //delete canvas()->popPen();
    delete canvas()->popBrush();

    //set up the display parameters
    // -- select a newly created font
    canvas()->backColor(GRAY);
    canvas()->setTextBackMode(ZTEXT_TRANSPARENT);

    r_ndx= DxBufNdx;

    for(tmp_ndx=0;tmp_ndx<N_DXMSGGS;tmp_ndx++) //loop through the number of

```

```

{
    //lines displayed in the window
    r_curr = r_ndx-tmp_ndx; //from most current to past
    if( (r_curr <= DxBufInp)
    &&(r_curr >=0 )){
        msg_curr[tmp_ndx] = DxBuf[r_curr];
    }else{
        msg_curr[tmp_ndx] = NULL;
    }//endif
} //endfor

for(tmp_ndx=0;tmp_ndx<N_DXMSG;tmp_ndx++) //loop through the number of
{
    //lines displayed in the window
    // the following outputs a message to the DXFile window
    if(msg_curr[tmp_ndx]!=NULL)
    {
        canvas()->text(x_mbase,
                                (y_mbase+(tmp_ndx*r_offset)),
                                msg_curr[tmp_ndx]);

    } //endif
} //endfor

delete canvas()->popFont();
// zpb_end
canvas()->unlock();
return 1;
}

//
// Frame Member Functions - DxWin
//
// Window Constructor
WDxWin::WDxWin(zMDIAppFrame *w, const char *title)
: zMDIChildFrame(w,ZNEW zSizer(zDialogUnit(0,0),
    zDialogUnit(310,130)),WS_CHILD|WS_THICKFRAME|WS_SYSMENU|WS_MINIMIZEBOX|WS_CAPTION, title) {
    // zpb_begin WDxWinConstructor2
    drwRt=1000L;
    drwTm=0L;
    msgRt=255L;
    msgTm=0L;
    #define DXSR 8L // this object wakes up at the fastest message freq.

    // zpb_end
    deleteOnClose(TRUE);
    backgroundColor(zColor(255,128,128));
    menu(ZNEW zMenu(this, zResId(IDM_DxWinDxMenu)));
    menu()->setCommand(this, (CommandProc)&WDxWin::cmdControlRefresh, IDM_CONTROLREFRESH);
    menu()->setCommand(this, (CommandProc)&WDxWin::cmdControlMsgRate, IDM_CONTROLMSGRATE);
    pDxTxtPane = ZNEW DxWinDxTxtPane(this, ZNEW zGravSizer(ZGRAV_MIDDLE,0,sizer()));
    pDxTxtPane->show();
    sizer()->update();
    // zpb_begin WDxWinConstructor1 OpenDlg
    char *types[6];
    types[0] = "All Files (*.*)", types[1] = "*.**";
    types[2] = "Text Files (*.txt)", types[3] = "*.txt";
    types[4] = types[5] = 0;

    zFileOpenForm *p = ZNEW zFileOpenForm(this,"Open File", 0, types);
    if (p->completed()) {
        // Use p->name() to retrieve filename
        char *fname;
        char *line;
        fstream fin;
        fname = p->name();
        fin.open(fname,ios::in);
        DxBufInp=0;
        line = new char[LBUFSZ+1];

```



```

fin.getline(line,LBUFSZ);
while( ( line[2] != ':')

                                ||(line[5] != ':')
                                ||(line[8] != ':') )
                                &&(DxBufInp < 20 )){

fin.getline(line,LBUFSZ);
DxBufInp++;
} //endwhile
DxBufInp=0;
DxBuf[DxBufInp] = line;
DxBufInp++;
line = new char[LBUFSZ+1];
char tbuf[64];
while( (fin.getline(line,LBUFSZ) )

                                &&(DxBufInp < DXBUFSZ )){

//strcpy(tbuf,line + 15, 4); //type of track
//int tbufndx;
//for(tbufndx=0;tbufndx<4;tbufndx++){
//
//                                tbuf[tbufndx] = *(line+15));
//} //endfor
if(*(line+16)) == '2'){ //verify valid message type
                                DxBuf[DxBufInp] = line;
                                DxBufInp++;
                                line = new char[LBUFSZ+1];
} //endif
} //endwhile
fin.close();
DxBufNdx=0; //Set buffer index to beginning of buffer
}
delete p;
//zpb_end

// zpb_begin WDXWinConstructor1

// this block of code finds a previous message that is "closest" to
// the current message. The index of this message is saved. Note that
// the previous message has an index to a previous message that it
// was closest to. Those two messages will be used to generate a
// "delta message" format. This block of code links the messages
// before the demo starts. After the algorithm is tested and verified
// the code will be moved to the location where it will run in realtime.
// int tmpndx;
// #define CCMAX 20 // compare last CCMAX messages from current
// for(tmpndx=CCMAX,tmpndx<DxBufInp,tmpndx++){ //start comparing after CCMAX
// char *lnptr = DxBuf[tmpndx]; //current line pointer
// int lnsz = strlen(lnptr); //size of current line
// int ccndx; //index into the messages
// for(ccndx=CCMAX;ccndx>0;ccndx--){ //last CCMAX messages to be compared
//
//                                char *strptr = DxBuf[tmpndx-CCMAX]; //previous message
//                                int strsz = strlen(strptr); //size of previous message
//                                int strmin; //min message length
//                                if(lnsz<strsz){
//                                strmin=lnsz; //current line is shortest
//                                } else {
//                                strmin=strsz; //previous line is shortest
//                                } //endif
//                                int strdel = abs(lnsz - strsz); //diff is counted as diff chars
//                                int slndx;
//                                for(slndx=0;slndx<strmin;slndx++){ //compare char for char
//                                if( (*strptr) != (*lnptr) ){ //if diff then inc delta
//                                strdel++;
//                                } //endif
//                                strptr++; //inc char location in each str
//                                lnptr++;
//                                } //endfor
// } //endfor
// } //endfor

```

```

        // zpb_end
        // zpb_begin WDXWinConstructor

        //Create the track display for this message stream
        ZNEW WTrkWin((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_TRKWIN)));
        pSQ->Slp(this, DXSR); //wake window every x ticks

        // zpb_end
        show();
    }

    WDXWin::~WDXWin() {
        // zpb_begin WDXWinDestructor!
        pSQ->Rm(this); //Remove this window from the sleep queue
        // zpb_end
    }

    //
    // Menu Item Selection Handlers
    //

    int WDXWin::cmdControlRefresh(zCommandEvt* ev) {
        DMsgRfsh* p=ZNEW DMsgRfsh(this, zResId(IDD_MsgRfsh));
        p->modal();
        if (p->completed()) {
            // zpb_begin WDXWinControlRefresh
            drwRt = (unsigned long) atol(p->_DRt);
            // zpb_end
        } else {
            // zpb_begin WDXWinControlRefreshCancel
            // zpb_end
        }
        delete p;
        return 0;
    }

    int WDXWin::cmdControlMsgRate(zCommandEvt* ev) {
        DMsgRate* p=ZNEW DMsgRate(this, zResId(IDD_MsgRate));
        p->modal();
        if (p->completed()) {
            // zpb_begin WDXWinControlMsgRate
            msgRt = (unsigned long) atol(p->_Mrt);
            // zpb_end
        } else {
            // zpb_begin WDXWinControlMsgRateCancel
            // zpb_end
        }
        delete p;
        return 0;
    }

    // zpb_begin WDXWinMemberFunctions

    int WDXWin::wake() {
        zDrawEvt *tmpEv;
        if(pSQ->stime >= drwTm){
            if(pTQ != NULL){
                if(pTQ->TQ != NULL){
                    int updndx;
                    for(updndx=0;updndx<pTQ->nTrks;updndx++){ //Refresh txt window
                        if(pTQ->TQ[updndx]->TxWn != NULL){
                            pTQ->TQ[updndx]->TxWn->rtrdrw();
                        }
                    }
                    if(pTQ->TQ[updndx]->AnWn != NULL){
                        pTQ->TQ[updndx]->AnWn->rtrdrw();
                    }
                }
            }
        }
    }

```

```

        } //endif
    } //endif
    } //endif
    pDxTxtPane->draw(tmpEv);
    drwTm = pSQ->systime + drwRt;
    } //endif
    if(pSQ->systime >= msgTm){
        DxBufNdx++;
        if(DxBufNdx>=DxBufInp){ //not end of buffer, i.e. index less than inputs
            DxBufNdx = 0; //loop back to beginning of message buffer
        } //endif

        pTQ->upd((char *)DxBuf[DxBufNdx]); //update the track display queue
                                          //with the new message
        msgTm = pSQ->systime + msgRt;
    } //endif
    return 0;
}

//
// Sleep Queue Member Functions
//
// constructor
SlpQ::SlpQ(){
    systime = GetCurrentTime();
    int ndx;
    for(ndx=0;ndx<NTASKS;ndx++){
        sq[ndx] = NULL;
        SlTm[ndx]=0;
    } //endfor ndx
} //end SlpQ()
//
// destructor
SlpQ::~SlpQ()
{ //end ~SlpQ()
//
// Periodic Execution Every Specified Number of Ticks
int SlpQ::Slp(zMDIChildFrame *wptr, unsigned long tcks){
    int ndx;
    for(ndx=0;ndx<NTASKS;ndx++){
        if( (sq[ndx] == NULL) ){
            sq[ndx] = wptr;
            SlTm[ndx]=tcks;
            WkTm[ndx]=systime+tcks;
            return 0;
        } //endif
    } //endfor ndx
    return 0;
} //end Slp()
//
// Check the Sleep Queue for Any Member Functions
int SlpQ::CkQ() {
    systime = GetCurrentTime();
    int sndx;
    for(sndx=0;sndx<NTASKS;sndx++){
        if( (sq[sndx] != NULL )
            &&(WkTm[sndx] <= systime)){
            WkTm[sndx]=systime+SlTm[sndx];
            sq[sndx]->wake();
            return 0;
        } //endif
    } //endfor ndx
    return 0;
} //end CkQ()
//
// Remove window from the queue

```

```

int SlpQ::Rm(zMDIChildFrame *wptr){
    int ndx;
    for(ndx=0;ndx<NTASKS;ndx++){
        if( (sq[ndx] == wptr)){
            sq[ndx] =NULL;
            SITm[ndx]=0;
            WkTm[ndx]=0;
            return 0;
        }
    }
    return 0;
}

// End Sleep Queue Member Functions

// The Following is a Single Thread Used for Application Multitasking

DWORD WINAPI ThreadFunc(LPVOID lpvThreadParm){
    DWORD dwResult = 0;

    #define SlpQRt 1L //this is the fastest rate any method can run
    int done = 0;
    while(done == 0){
        done = pSQ->CkQQ;
        Sleep(SlpQRt);
    }

    return(dwResult);
}

// zpb_end

// Pane Member Functions - TrkWinTrkPane
TrkWinTrkPane::TrkWinTrkPane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD|WS_BORDER) {

    // zpb_begin pTrkWinTrkPaneConstructor1

    //this is where we determine which track to display
    bmppha = new zBitmap(canvas(), "sym/diamup.bmp" );
    bmpphs = new zBitmap(canvas(), "sym/diamdwn.bmp" );
    bmpphsf = new zBitmap(canvas(), "sym/diam.bmp" );
    bmpukna = new zBitmap(canvas(), "sym/sqrup.bmp" );
    bmpukns = new zBitmap(canvas(), "sym/sqrdwn.bmp" );
    bmpuknsf = new zBitmap(canvas(), "sym/sqr.bmp" );
    bmpfa = new zBitmap(canvas(), "sym/circleup.bmp" );
    bmpfs = new zBitmap(canvas(), "sym/circledwn.bmp");
    bmpfsf = new zBitmap(canvas(), "sym/circle.bmp" );
    // zpb_end
    show();
}

int TrkWinTrkPane::mouseButtonDown(zMouseClickedEvt* ev) {
    if (ev->isButton(1)) { // Left Button Pressed
        // zpb_begin TrkWinTrkPaneLButtonDown
        int DTndx = 0;
        zPoint mpos;
        mpos = ev->pos(); //gives us the position when clicked
        int mousex = mpos.x();
        int mousey = mpos.y();
        int closex = pTQ->TQ[DTndx]->xpos;
        int closey = pTQ->TQ[DTndx]->ypos;
        pTQ->DTndx = DTndx;
        for(DTndx=1;DTndx<pTQ->nTrks;DTndx++){

```

```

    if( ( abs(pTQ->TQ[DTndx]->xpos - mousex)
        < abs( closex - mousex))
        &&( abs(pTQ->TQ[DTndx]->ypos - mousey)
            < abs( closey - mousey))) {
        closex = pTQ->TQ[DTndx]->xpos;
        closey = pTQ->TQ[DTndx]->ypos;
        pTQ->DTndx = DTndx;    //update the DTInfo() ndx
    } //endif
} //endfor

DTInfo* p=ZNEW DTInfo(this, zResId(IDD_TInfo));
p->modal();
if (p->completed()) {
    // hook for later code
    pTQ->TQ[pTQ->DTndx]->distype = p->UpdAtt(pTQ->TQ[pTQ->DTndx]->dTypStr);
} //endif
delete p;

// zpb_end
}
else
if (ev->isButton(2)) { // Right Button Pressed
// zpb_begin TrkWinTrkPaneRButtonDown
int NTndx = 0;
zPoint mpos;
mpos = ev->pos(); //gives us the position when clicked
int mousex = mpos.x();
int mousey = mpos.y();
int closex = pTQ->TQ[0]->xpos;
int closey = pTQ->TQ[0]->ypos;
pTQ->newTrkTxt = 0;
for(NTndx=1; NTndx<pTQ->nTrks; NTndx++){
    if( ( abs(pTQ->TQ[NTndx]->xpos - mousex)
        < abs( closex - mousex))
        &&( abs(pTQ->TQ[NTndx]->ypos - mousey)
            < abs( closey - mousey))) {
        closex = pTQ->TQ[NTndx]->xpos;
        closey = pTQ->TQ[NTndx]->ypos;
        pTQ->newTrkTxt = NTndx;    //update the TrkTxt() ndx
    } //endif
} //endfor

const char trkmsg[256] = "Track Analysis";
//strcat(trkmsg, pTQ->TQ[NTndx]->numStr);
if(pTQ->TQ[pTQ->newTrkTxt] != NULL){ //Track must exist
if(pTQ->TQ[pTQ->newTrkTxt]->TxWn == NULL){ //Only one window per track
    ZNEW WTrkTxt((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), trkmsg);
} //endif
} //endif

// zpb_end
}
return 0;
}

int TrkWinTrkPane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin TrkWinTrkPaneSize
    // zpb_end
    return zWindow::size(ev);
}

int TrkWinTrkPane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin TrkPaneDraw
    //local constants

```

```
//local variables
```

```
//clear the window pane
```

```
zRect DxTxtArea;
```

```
canvas()->setVisible(DxTxtArea);
```

```
canvas()->pushPen(new zPen(zColor(GREEN), Solid, 5));
```

```
canvas()->pushBrush(new zBrush(DarkGrayBrush));
```

```
canvas()->pushBrush(new zBrush(zColor(0,0,0)));
```

```
canvas()->rectangle(DxTxtArea);
```

```
delete canvas()->popPen();
```

```
delete canvas()->popBrush();
```

```
//set up the display parameters
```

```
// -- select a newly created font
```

```
//canvas()->backColor(zColor(0,0,0));
```

```
canvas()->setTextBackMode(ZTEXT_TRANSPARENT);
```

```
canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));
```

```
//draw a line - will use the pen
```

```
int ndx;
```

```
for(ndx=0;ndx<pTQ->nTrks;ndx++){
```

```
    canvas()->textColor(pTQ->TQ[ndx]->clr);
```

```
    canvas()->pushPen(new zPen(pTQ->TQ[ndx]->clr,Solid,1));
```

```
    float xtmp,ytmp;
```

```
    int x,y;
```

```
    #define WINDLT 3
```

```
    xtmp = ((float)DxTxtArea.width())
```

```
        * ((pTQ->TQ[ndx]->xRaw - pTQ->minlt)/pTQ->dellt);
```

```
    x = DxTxtArea.left() + (int)xtmp;
```

```
    if(x<=(DxTxtArea.left() + WINDLT)){
```

```
        x=DxTxtArea.left() + WINDLT;
```

```
    }//endif check lower bound
```

```
    if(x>=(DxTxtArea.right()-WINDLT)){
```

```
        x=DxTxtArea.right() - WINDLT;
```

```
    }//check upper bound
```

```
    ytmp = ((float)DxTxtArea.height())
```

```
        * ((pTQ->TQ[ndx]->yRaw - pTQ->minlg)/pTQ->dellg);
```

```
    y = DxTxtArea.top() + (int)ytmp; //new y for display
```

```
    if(y<=(DxTxtArea.top() + WINDLT)){
```

```
        y=DxTxtArea.top() + WINDLT; //clip to top of window
```

```
    }//endif check lower bound
```

```
    if(y>=(DxTxtArea.bottom()-WINDLT)){
```

```
        y=DxTxtArea.bottom() - WINDLT; //clip to bottom of window
```

```
    }//check upper bound
```

```
    switch(pTQ->TQ[ndx]->distype) {
```

```
        case HOSTILE_AIR:
```

```
            canvas()->bitmap(bmphaz,zPoint(x-10,y-20), SRCCOPY);
```

```
            canvas()->textColor(zColor(255,0,0));
```

```
            canvas()->text(x-11, y,pTQ->TQ[ndx]->numStr);
```

```
            if( (pTQ->TQ[ndx]->xpos != 0)
```

```
                ||(pTQ->TQ[ndx]->ypos != 0)){
```

```
                canvas()->moveTo(x,y);
```

```
                canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
```

```
            }//endif
```

```
            pTQ->TQ[ndx]->xpos = x;
```

```
            pTQ->TQ[ndx]->ypos = y;
```

```
            break;
```

```
        case HOSTILE_SUB:
```

```
            canvas()->bitmap(bmphs,zPoint(x-10,y), SRCCOPY);
```

```
            canvas()->textColor(zColor(255,0,0));
```

```
            canvas()->text(x-11, y+20,pTQ->TQ[ndx]->numStr);
```

```
            if( (pTQ->TQ[ndx]->xpos != 0)
```

```
                ||(pTQ->TQ[ndx]->ypos != 0)){
```

```
                canvas()->moveTo(x,y);
```

```
                canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
```

```

    }//endif
    pTQ->TQ[ndx]->xpos = x;
    pTQ->TQ[ndx]->ypos = y;
    break;
case HOSTILE_SURFACE:
    canvas()->bitmap(bmphsf,zPoint(x-12,y-12), SRCCOPY);
    canvas()->textColor(zColor(255,0,0));
    canvas()->text(x-12, y+11,pTQ->TQ[ndx]->numStr);
    if( (pTQ->TQ[ndx]->xpos != 0)
        ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
    }//endif
    pTQ->TQ[ndx]->xpos = x;
    pTQ->TQ[ndx]->ypos = y;
    break;
case UNKNOWN_AIR:
    canvas()->bitmap(bmpukna,zPoint(x-10,y-20), SRCCOPY);
    canvas()->textColor(zColor(255,255,255));
    canvas()->text(x-12, y,pTQ->TQ[ndx]->numStr);
    if( (pTQ->TQ[ndx]->xpos != 0)
        ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
    }//endif
    pTQ->TQ[ndx]->xpos = x;
    pTQ->TQ[ndx]->ypos = y;
    break;
case UNKNOWN_SUB:
    canvas()->bitmap(bmpukns,zPoint(x-10,y), SRCCOPY);
    canvas()->textColor(zColor(255,255,255));
    canvas()->text(x-12, y+20,pTQ->TQ[ndx]->numStr);
    if( (pTQ->TQ[ndx]->xpos != 0)
        ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
    }//endif
    pTQ->TQ[ndx]->xpos = x;
    pTQ->TQ[ndx]->ypos = y;
    break;
case UNKNOWN_SURFACE:
    canvas()->bitmap(bmpuknsf,zPoint(x-10,y-10), SRCCOPY);
    canvas()->textColor(zColor(255,255,255));
    canvas()->text(x-11, y+10,pTQ->TQ[ndx]->numStr);
    if( (pTQ->TQ[ndx]->xpos != 0)
        ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
    }//endif
    pTQ->TQ[ndx]->xpos = x;
    pTQ->TQ[ndx]->ypos = y;
    break;
case FRIENDLY_AIR:
    canvas()->bitmap(bmpfa,zPoint(x-8,y-17), SRCCOPY);
    canvas()->textColor(zColor(0,255,255));
    canvas()->text(x-9, y+1,pTQ->TQ[ndx]->numStr);
    if( (pTQ->TQ[ndx]->xpos != 0)
        ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
    }//endif
    pTQ->TQ[ndx]->xpos = x;
    pTQ->TQ[ndx]->ypos = y;
    break;
case FRIENDLY_SUB:
    canvas()->bitmap(bmpfs,zPoint(x-8,y), SRCCOPY);
    canvas()->textColor(zColor(0,255,255));

```

```

        canvas()->text(x-9, y+18,pTQ->TQ[ndx]->numStr);
        if (pTQ->TQ[ndx]->xpos != 0)
            ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
        }//endif
        pTQ->TQ[ndx]->xpos = x;
        pTQ->TQ[ndx]->ypos = y;
        break;
    case FRIENDLY_SURFACE:
        canvas()->bitmap(bmpfsf,zPoint(x-11,y-11), SRCCOPY);
        canvas()->textColor(zColor(0,255,255));
        canvas()->text(x-11, y+12,pTQ->TQ[ndx]->numStr);
        if (pTQ->TQ[ndx]->xpos != 0)
            ||(pTQ->TQ[ndx]->ypos != 0)){
        canvas()->moveTo(x,y);
        canvas()->lineTo(pTQ->TQ[ndx]->xpos,pTQ->TQ[ndx]->ypos);
        }//endif
        pTQ->TQ[ndx]->xpos = x;
        pTQ->TQ[ndx]->ypos = y;
        break;

}; //end case statement
//pTQ->TQ[ndx]->distype++;
//if (pTQ->TQ[ndx]->distype > 8)pTQ->TQ[ndx]->distype = 0;
//canvas()->text(x, y,pTQ->TQ[ndx]->numStr);

delete canvas()->popPen();
} //endfor ndx

//canvas()->textColor(BLACK);
delete canvas()->popFont();

// zpb_end
canvas()->unlock();
return 1;
}

//
// Frame Member Functions - TrkWin
//
// Window Constructor
WTrkWin::WTrkWin(zMDIAppFrame *w, const char *title)
: zMDIChildFrame(w,ZNEW zSizer(zDialogUnit(310,0), zDialogUnit(200,155)),zSTDFRAME, title) {
    // zpb_begin WTrkWinConstructor2

    drwRt=500L;
    drwTm=0L;
    #define TRSR 10L // this object wakes up at the fastest message freq.
    // zpb_end
    deleteOnClose(TRUE);
    menu(ZNEW zMenu(this, zResId(IDM_TrkWinTrkMenu)));
    menu()->setCommand(this, (CommandProc)&WTrkWin::cmdControlAutoSet, IDM_CONTROLAUTOSSET);
    menu()->setCommand(this, (CommandProc)&WTrkWin::cmdControlRefresh, IDM_CONTROLREFRESH);
    menu()->setCommand(this, (CommandProc)&WTrkWin::cmdControlSetup, IDM_CONTROLSETUP);
    menu()->setCommand(this, (CommandProc)&WTrkWin::cmdDisplayTrack, IDM_DISPLAYTRACK);
    pTrkPane = ZNEW TrkWinTrkPane(this, ZNEW zGravSizer(ZGRAV_MIDDLE,0,sizer()));
    pTrkPane->show();
    sizer()->update();
    // zpb_begin WTrkWinConstructor
    //pStaticIcon1->backgroundColor(zColor(64,64,64));
    pSQ->Slp(this, TRSR); //wake window every x ticks
    // zpb_end
    show();
}

WTrkWin::~WTrkWin() {
    // zpb_begin WTrkWinDestructor1

```



```

        pSQ->Rm(this); //Remove this window from the sleep queue
        pTQ->~TrkQ();
        // zpb_end
    }

//
// Menu Item Selection Handlers
//

int WTrkWin::cmdControlAutoSet(zCommandEvt* ev) {
    // zpb_begin WTrkWinControlAutoSet
    pTQ->rescale();
    // zpb_end
    return 0;
}

int WTrkWin::cmdControlRefresh(zCommandEvt* ev) {
    DTrkRfsh* p=ZNEW DTrkRfsh(this, zResId(IDD_TrkRfsh));
    p->modal();
    if (p->completed()) {
        // zpb_begin WTrkWinControlRefresh
        drwRt = (unsigned long) atol(p->_DRt);
        // zpb_end
    } else {
        // zpb_begin WTrkWinControlRefreshCancel
        // zpb_end
    }
    delete p;
    return 0;
}

int WTrkWin::cmdControlSetup(zCommandEvt* ev) {
    DTkSet* p=ZNEW DTkSet(this, zResId(IDD_TkSet));
    p->modal();
    if (p->completed()) {
        // zpb_begin WTrkWinControlSetup
        pTQ->minlt = p->_laMn; //this is for op area field of view
        pTQ->maxlt = p->_laMx;
        pTQ->minlg = p->_lgMn;
        pTQ->maxlg = p->_lgMx;
        pTQ->dellt = pTQ->maxlt - pTQ->minlt;
        pTQ->delld = pTQ->maxlg - pTQ->minlg;
        // zpb_end
    } else {
        // zpb_begin WTrkWinControlSetupCancel
        // zpb_end
    }
    delete p;
    return 0;
}

int WTrkWin::cmdDisplayTrack(zCommandEvt* ev) {
    ZNEW WTrkTxt((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_TRKTXT)));
    // zpb_begin WTrkWinDisplayTrack
    // zpb_end
    return 0;
}

// zpb_begin WTrkWinMemberFunctions
int WTrkWin::wake(){
    zDrawEvt *tmpEv;
    if(pSQ->systime >= drwTm){
        pTrkPane->draw(tmpEv);
        drwTm = pSQ->systime + drwRt;
    }//endif
    return 0;
} //end wake()

```

```

// zpb_end

// Pane Member Functions - SymWinSymPane
SymWinSymPane::SymWinSymPane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD) {

    // zpb_begin pSymWinSymPaneConstructor1
    // zpb_end
    show();
}

int SymWinSymPane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin SymWinSymPaneSize
    // zpb_end
    return zWindow::size(ev);
}

int SymWinSymPane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin SymPaneDraw
    //set up the display parameters
    canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));

    //clear the window pane
    zRect DxTxtArea;
    canvas()->setVisible(DxTxtArea);
    canvas()->rectangle(DxTxtArea);

    canvas()->text(10,
                                     10,
                                     "hello world");

    delete canvas()->popFont();
    // zpb_end
    canvas()->unlock();
    return 1;
}

//
// Frame Member Functions - SymWin
//
// Window Constructor
WSymWin::WSymWin(zMDIAppFrame *w, const char *title)
: zMDIChildFrame(w,ZNEW zSizer(zDialogUnit(310,0),
                                zDialogUnit(200,255)),WS_CHILD|WS_THICKFRAME|WS_SYSMENU|WS_MINIMIZEBOX|WS_CAPTION, title) {
    // zpb_begin WSymWinConstructor2

    // zpb_end
    deleteOnClose(TRUE);
    menu(ZNEW zMenu(this, zResId(IDM_SymWinSymMenu)));
    menu()->setCommand(this, (CommandProc)&WSymWin::cmdControlRefresh, IDM_CONTROLREFRESH);
    pSymPane = ZNEW SymWinSymPane(this, ZNEW zGravSizer(ZGRAV_MIDDLE,0,sizer()));
    pSymPane->show();
    sizer()->update();
    // zpb_begin WSymWinConstructor
    // zpb_end
    show();
}

WSymWin::~WSymWin() {
    // zpb_begin WSymWinDestructor1
    // zpb_end
}

//

```

```

// Menu Item Selection Handlers
//

int WSymWin::cmdControlRefresh(zCommandEvt* ev) {
    DSymRfsh* p=ZNEW DSymRfsh(this, zResId(IDD_SymRfsh));
    p->modal();
    if (p->completed()) {
        // zpb_begin WSymWinControlRefresh
        // zpb_end
    } else {
        // zpb_begin WSymWinControlRefreshCancel
        // zpb_end
    }
    delete p;
    return 0;
}

// zpb_begin WSymWinMemberFunctions
// zpb_end

// Pane Member Functions - DelWinSymPane
DelWinSymPane::DelWinSymPane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD) {

    // zpb_begin pDelWinSymPaneConstructor1
    // zpb_end
    show();
}

int DelWinSymPane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin DelWinSymPaneSize
    // zpb_end
    return zWindow::size(ev);
}

int DelWinSymPane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin SymPaneDraw
    //set up the display parameters
    canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));

    //clear the window pane
    zRect DxTxtArea;
    canvas()->setVisible(DxTxtArea);
    canvas()->rectangle(DxTxtArea);

    canvas()->text(10,
                                     10,
                                     "hello world");

    delete canvas()->popFont();
    // zpb_end
    canvas()->unlock();
    return 1;
}

//
// Frame Member Functions - DelWin
//
// Window Constructor
WDelWin::WDelWin(zMDIAppFrame *w, const char *title)
: zMDIChildFrame(w,ZNEW zSizer(zDialogUnit(310,255),
    zDialogUnit(200,112)),WS_CHILD|WS_THICKFRAME|WS_SYSMENU|WS_MINIMIZEBOX|WS_CAPTION, title) {
    // zpb_begin WDelWinConstructor2
    // zpb_end
}

```

```

        deleteOnClose(TRUE);
        menu(ZNEW zMenu(this, zResId(IDM_DelWinSymMenu)));
        menu()->setCommand(this, (CommandProc)&WDelWin::cmdControlRefresh, IDM_CONTROLREFRESH);
        pSymPane = ZNEW DelWinSymPane(this, ZNEW zGravSizer(ZGRAV_MIDDLE,0,sizer()));
        pSymPane->show();
        sizer()->update();
        // zpb_begin WDelWinConstructor
        // zpb_end
        show();
    }

WDelWin::~WDelWin() {
    // zpb_begin WDelWinDestructor1
    // zpb_end
}

//
// Menu Item Selection Handlers
//

int WDelWin::cmdControlRefresh(zCommandEvt* ev) {
    DSymRfsh* p=ZNEW DSymRfsh(this, zResId(IDD_SymRfsh));
    p->modal();
    if (p->completed()) {
        // zpb_begin WDelWinControlRefresh
        // zpb_end
    } else {
        // zpb_begin WDelWinControlRefreshCancel
        // zpb_end
    }
    delete p;
    return 0;
}

// zpb_begin WDelWinMemberFunctions
// zpb_end

// Pane Member Functions - TrkTxtTkTxPane
TrkTxtTkTxPane::TrkTxtTkTxPane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD) {

    // zpb_begin pTrkTxtTkTxPaneConstructor1
    TrkNdx = pTQ->newTrkTxt; //set the track pointer to the New Track
    // zpb_end
    show();
}

int TrkTxtTkTxPane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin TrkTxtTkTxPaneSize
    // zpb_end
    return zWindow::size(ev);
}

int TrkTxtTkTxPane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin TkTxPaneDraw

//local constants
#define N_TKMSGS 24 // total number of messages displayed on the pane
#define X_MB 3 // x coordinate for base (first row) of display messages
#define Y_MB 10 // y coordinate for base (first row) of display messages
#define R_OFF 15 //offset for spacing between rows

//local variables
int tmp_ndx; //temporary index
int lbufNdx;
int r_ndx; //row index for looping through the active messages currently displayed

```

```

int r_curr;                                     //actual index into the buffer of messages

//char *msg_curr[N_TKMSGGS];

//set up the display parameters
//canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));
canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));

//clear the window pane
zRect DxTxtArea;
canvas()->setVisible(DxTxtArea);
//canvas()->pushPen(new zPen(zColor(GREEN), Solid, 5));
canvas()->pushBrush(new zBrush(zColor(255,255,255)));
canvas()->rectangle(DxTxtArea);
//delete canvas()->popPen();
delete canvas()->popBrush();

//set up the display parameters
// - select a newly created font
//canvas()->backColor(GRAY);
canvas()->setTextBackMode(ZTEXT_TRANSPARENT);

int nMsgs;
if(pTQ->TQ[TrkNdx]->MQndx < N_TKMSGGS){
    nMsgs = pTQ->TQ[TrkNdx]->MQndx;
    r_ndx = 0;
} else {
    nMsgs = N_TKMSGGS;
    r_ndx = pTQ->TQ[TrkNdx]->MQndx - N_TKMSGGS;
} //endif

zDimension txtDim; //used to move the draw origin as chars output
int xdelta;        //number of pixels for chars draw on a line
canvas()->textColor(zColor(0,0,0)); //black text
char *ptc1;
char *ptc0;
char *ptcm1;
int numchrs;
int ndx;
int dsum;
for(tmp_ndx=(nMsgs-1);tmp_ndx>=0;tmp_ndx--) //loop through the number of
{
    //lines displayed in the window
    r_curr = r_ndx+tmp_ndx;

    zPoint p( X_MB,
               (Y_MB+(((nMsgs-1)-tmp_ndx)*R_OFF)));

    int scnt[LBUFSZ];
    xdelta = 0;
    if((nMsgs>1)&&(r_curr>0)){
        ptc1 = pTQ->TQ[TrkNdx]->MQ[r_curr]; //get pointer to line buffers
        ptc0 = pTQ->TQ[TrkNdx]->MQ[r_curr-1];
        numchrs = strlen(ptc1);
        dsum = 0;
        int ccnt;
        for(ccnt=0;(ccnt<numchrs)&&(ccnt<LBUFSZ);ccnt++){
            if( ((*ptc1) != (*ptc0))){
                scnt[ccnt] = 1;
                pTQ->TQ[TrkNdx]->CurDel[ccnt] = 1;
                dsum++; //this counts number of chars not same
            } else {
                scnt[ccnt] = 0;
                pTQ->TQ[TrkNdx]->CurDel[ccnt] = 0;
            } //endif
            ptc1++;
            ptc0++;
        } //endfor
        if(nMsgs>2)&&(r_curr>1){ //test for "delta messaging"

```

```

        ptc0 = pTQ->TQ[TrkNdx]->MQ[r_curr-1]; //reset pointer to line buffers
        ptc1 = pTQ->TQ[TrkNdx]->MQ[r_curr-2];
        for(ccnt=0;(ccnt<numchrs)&&(ccnt<LBUFSZ);ccnt++){
            if((*ptc0) != (*ptc1)){
                pTQ->TQ[TrkNdx]->OldDel[ccnt] = 1;
            }else{
                pTQ->TQ[TrkNdx]->OldDel[ccnt] = 0;
            }
            ptc0++;
            ptc1++;
        }
    }
    ptc1 = pTQ->TQ[TrkNdx]->MQ[r_curr]; //reset pointer to line buffer
    int pcnt;
    ccnt=0;
    while((ccnt<numchrs)&&(ccnt<LBUFSZ)){
        pcnt = 0;
        while(scnt[ccnt]==0){
            pcnt++;ccnt++;
        }
        if((pcnt>0)&&(ccnt<=numchrs)){
            canvas()->textColor(zColor(0,0,0)); //black text if same
            zPoint ddc(p.x() + xdelta,p.y());
            canvas()->text(ddc,ptc1,pcnt);
            txtDim = canvas()->getTextDim(ptc1,pcnt);
            xdelta += (txtDim.width()-1);
            ptc1 += pcnt;
        }
        if(ccnt<numchrs){
            canvas()->textColor(zColor(255,0,0)); //red text if different
            zPoint ddc(p.x() + xdelta,p.y());
            canvas()->text(ddc,ptc1,1);
            txtDim = canvas()->getTextDim(ptc1,1);
            xdelta += (txtDim.width()-1);
            ccnt++;
            ptc1++;
        }
    }

    int dmsg = 0; // test to see if we have a delta message
    int tmdel = strlen(pTQ->TQ[TrkNdx]->timStr) + 1;
    for(ccnt=tmdel;
        ((ccnt<numchrs)&&(ccnt<LBUFSZ));
        ccnt++){
        if( ( pTQ->TQ[TrkNdx]->CurDel[ccnt] == 1 ) //we have a new char
            &&( pTQ->TQ[TrkNdx]->OldDel[ccnt] == 0 )){ //no delta message
            dmsg++;
        }
    }

    xdelta = 700;
    ptc1 = pTQ->TQ[TrkNdx]->MQ[r_curr]; //reset pointer to curr buffer
    ptc1 += tmdel;
    if(dmsg == 0){ //we have a delta message
        char dm[8] = "!!!";
        canvas()->textColor(zColor(0,0,255)); //blue text
        zPoint ddc(p.x() + xdelta,p.y());
        canvas()->text(ddc,dm,1);
        txtDim = canvas()->getTextDim(dm,1);
        xdelta += (txtDim.width()-1);
        for(ccnt=tmdel;
            (ccnt<numchrs)&&(ccnt<LBUFSZ);
            ccnt++){
            if( pTQ->TQ[TrkNdx]->OldDel[ccnt] == 1 ){ //print the delta char
                if(ccnt<numchrs){
                    canvas()->textColor(zColor(0,0,255)); //blue text
                }
            }
        }
    }
}

```

```

        zPoint ddc(p.x() + xdelta,p.y());
        canvas()->text(ddc,ptc1,1);
        txtDim = canvas()->getTextDim(ptc1,1);
        xdelta += (txtDim.width()-1);
    }//endif
} //endif
ptc1++;
} //endfor

} else {

        char dm[8] = "???";
        canvas()->textColor(zColor(0,0,0)); //black text
        zPoint ddc(p.x() + xdelta,p.y());
        canvas()->text(ddc,dm,1);
        txtDim = canvas()->getTextDim(dm,1);
        xdelta += (txtDim.width()-1);
    } //endif

    //the following calculates values needed for the analysis window
    numchrs = 100; //this adjusts for extra constant chars in the strings
    if(numchrs<0){numchrs=0;}
    if(numchrs<dsum){dsum=numchrs;}
    pTQ->TQ[TrkNdx]->cCnt[r_curr] = numchrs; //store total number of chars
    pTQ->TQ[TrkNdx]->dCnt[r_curr] = dsum;
    if(numchrs>0){
        pTQ->TQ[TrkNdx]->csRatio[r_curr] = ((float)dsum/(float)numchrs);
    } else {
        pTQ->TQ[TrkNdx]->csRatio[r_curr] = 0.0;
    } //endif
} else {
        canvas()->textColor(zColor(255,0,0)); //red text if first message
        canvas()->text(p,
            pTQ->TQ[TrkNdx]->MQ[r_curr]);
    } //endif
    canvas()->textColor(zColor(0,0,0)); //black text is default

} //endfor

delete canvas()->popFont();

// zpb_end
canvas()->unlock();
return 1;
}

//
// Frame Member Functions - TrkTxt
//
// Window Constructor
WTrkTxt::WTrkTxt(zMDIAppFrame *w, const char *title)
: zMDIChildFrame(w,ZNEW zSizer(zDialogUnit(-2,97),
    zDialogUnit(310,167)),WS_CHILD|WS_THICKFRAME|WS_SYSMENU|WS_MINIMIZEBOX|WS_CAPTION, title) {
    // zpb_begin WTrkTxtConstructor2
    if(pTQ->nTrks>0){
        zRect wsz;
        zCoOrd xd;
        zCoOrd yd;
        getExterior(wsz);
        xd = pTQ->xdTrkTxt;
        yd = pTQ->yTrkTxt;
        zPoint tmppt(xd,yd);
        wsz+=tmppt;
        move(wsz);
        pTQ->xdTrkTxt = 5;
    }
}

```

```

        pTQ->ydTrkTxt += 50;

    } //endif
    // zpb_end
    deleteOnClose(TRUE);
    backgroundColor(zColor(0,255,255));
    menu(ZNEW zMenu(this, zResId(IDM_TrkTxtTkTxMenu)));
    menu()->setCommand(this, (CommandProc)& WTrkTxt::cmdAnalysisThroughput, IDM_ANALYSISTHROUGHPUT);
    menu()->setCommand(this, (CommandProc)& WTrkTxt::cmdControlRefresh, IDM_CONTROLREFRESH);
    pTkTxPane = ZNEW TrkTxtTkTxPane(this, ZNEW zGravSizer(ZGRAV_MIDDLE,0,sizer()));
    pTkTxPane->show();
    sizer()->update();
    // zpb_begin WTrkTxtConstructor
    pTQ->TQ[pTQ->newTrkTxt]->TxWn = this; //this is a message text display window
    pTQ->newTrkAn = pTkTxPane->TrkNdx;
    ZNEW WDatAn((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_DATAN)));
    // zpb_end
    show();
}

WTrkTxt::~WTrkTxt() {
    // zpb_begin WTrkTxtDestructor1
    pTQ->TQ[pTkTxPane->TrkNdx]->TxWn = NULL; //this is a message text display window
    pTkTxPane->TrkNdx = 0;
    // zpb_end
}

//
// Menu Item Selection Handlers
//

int WTrkTxt::cmdAnalysisThroughput(zCommandEvt* ev) {
    ZNEW WDatAn((zMDIAppFrame*)zAppGetAppVar(app)->rootWindow(), zString(zResId(IDS_DATAN)));
    // zpb_begin WTrkTxtAnalysisThroughput
    //pTQ->newTrkAn = pTkTxPane->TrkNdx;
    // zpb_end
    return 0;
}

int WTrkTxt::cmdControlRefresh(zCommandEvt* ev) {
    DSymRfsh* p=ZNEW DSymRfsh(this, zResId(IDD_SymRfsh));
    p->modal();
    if (p->completed()) {
        // zpb_begin WTrkTxtControlRefresh
        // zpb_end
    } else {
        // zpb_begin WTrkTxtControlRefreshCancel
        // zpb_end
    }
    delete p;
    return 0;
}

// zpb_begin WTrkTxtMemberFunctions

int WTrkTxt::rtrdrw() {
    zDrawEvt *dummyEv;
    pTkTxPane->draw(dummyEv); //the draw routine does not use this input param
                                // zApp uses this as an event token for its
                                // own dispatching.
    return 0;
} //end rtrdrw()

// zpb_end

// Pane Member Functions - DatAnDatAnPane

```



```

DatAnDatAnPane::DatAnDatAnPane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD) {

    // zpb_begin pDatAnDatAnPaneConstructor1
    TrkNdx = pTQ->newTrkAn;
    // zpb_end
    show();
}

int DatAnDatAnPane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin DatAnDatAnPaneSize
    // zpb_end
    return zWindow::size(ev);
}

int DatAnDatAnPane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin DatAnPaneDraw

//local constants
#define N_TKMSGS                24    // total number of messages displayed on the pane
#define X_MB 3    // x coordinate for base (first row) of display messages
#define Y_MB 10    // y coordinate for base (first row) of display messages
#define R_OFF 15    //offset for spacing between rows

//local variables
int tmp_ndx;                //temporary index
int lbufNdx;
int r_ndx;                //row index for looping through the active messages currently displayed
int r_curr;                //actual index into the buffer of messages

//char *msg_curr[N_TKMSGS];

//set up the display parameters
//canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));
//canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));

//clear the window pane
zRect DxAnArea;
canvas()->setVisible(DxAnArea);
//canvas()->pushPen(new zPen(zColor(GREEN), Solid, 5));
//canvas()->pushBrush(new zBrush(LiteGrayBrush));
canvas()->rectangle(DxAnArea);
//delete canvas()->popPen();
delete canvas()->popBrush();

canvas()->setTextBackMode(ZTEXT_TRANSPARENT);

int nMsgs;
if(pTQ->TQ[TrkNdx]->MQndx < N_TKMSGS){
    nMsgs = pTQ->TQ[TrkNdx]->MQndx;
    r_ndx = 0;
} else {
    nMsgs = N_TKMSGS;
    r_ndx = pTQ->TQ[TrkNdx]->MQndx - N_TKMSGS;
}

char tbuf[256];
char tmp[16];
ostrstream tmpstr(tbuf,strlen(tbuf));

int x_margin;
x_margin = int (0.05*((float)DxAnArea.width()));
int y_margin;
y_margin = int (0.125*((float)DxAnArea.height()));
int c_off;
c_off = int (0.9*((float)DxAnArea.width())/N_TKMSGS);
int maxchrs = 30;

```

```

int y_unit;
y_unit = (DxAnArea.height() - (2*y_margin))/maxchrs;
canvas()->text(5,

5,
pTQ->TQ[TrkNdx]->numStr);

canvas()->text(50,

5,
pTQ->TQ[TrkNdx]->timStr);

for(tmp_ndx=(nMsgs-1);tmp_ndx>=0;tmp_ndx--) //loop through the number of
{
//lines displayed in the window
r_curr = r_ndx+tmp_ndx;

if((nMsgs>1)&&(r_curr>0)){
canvas()->pushPen(new zPen(zColor(GREEN), Solid, 5));
canvas()->moveTo(x_margin+(((nMsgs-1)-tmp_ndx)*c_off)),
DxAnArea.bottom() - y_margin );

//tmpstr << dec << pTQ->TQ[TrkNdx]->dCnt[r_curr]<<"0";
//strcpy(tmp,tbuf);
//int tbufndx = strlen(tmp)-1;
//tmp[tbufndx]='\0';

canvas()->lineTo(x_margin+(((nMsgs-1)-tmp_ndx)*c_off)),
( (DxAnArea.bottom() - y_margin )
-(y_unit * pTQ->TQ[TrkNdx]->cCnt[r_curr])));

delete canvas()->popPen();
canvas()->pushPen(new zPen(zColor(RED), Solid, 5));
canvas()->moveTo(x_margin+(((nMsgs-1)-tmp_ndx)*c_off)),
DxAnArea.bottom() - y_margin );

canvas()->lineTo(x_margin+(((nMsgs-1)-tmp_ndx)*c_off)),
( (DxAnArea.bottom() - y_margin )
-(y_unit*pTQ->TQ[TrkNdx]->dCnt[r_curr])));

delete canvas()->popPen();

//pTQ->TQ[TrkNdx]->cCnt[r_curr]; //store total number of chars
//pTQ->TQ[TrkNdx]->dCnt[r_curr] = dsum;
//if(numchrs>0){
//pTQ->TQ[TrkNdx]->csRatio[r_curr] = ((float)dsum/(float)numchrs);
//}else{
//
//pTQ->TQ[TrkNdx]->csRatio[r_curr] = 0.0;
//}
//endif
//this is the last message
//canvas()->textColor(zColor(255,0,0)); //red text if first message
//canvas()->text(p,
//
// "Insufficient Data for Analysis");
//endif
canvas()->textColor(zColor(0,0,0)); //black text is default

}

delete canvas()->popFont();
// zpb_end
canvas()->unlock();
return 1;
}

//
// Frame Member Functions - DatAn
//
// Window Constructor
WDatAn::WDatAn(zMDIAppFrame *w, const char *title)
: zMDIChildFrame(w,ZNEW zSizer(zDialogUnit(310,155),
zDialogUnit(100,100)),WS_CHILD|WS_THICKFRAME|WS_SYSMENU|WS_MINIMIZEBOX|WS_CAPTION, title) {
// zpb_begin WDatAnConstructor2
// zpb_end
deleteOnClose(TRUE);
backgroundColor(zColor(0,255,255));

```

```

        menu(ZNEW zMenu(this, zResId(IDM_DatAnDatAnMenu)));
        menu()->setCommand(this, (CommandProc)&WDatAn::cmdControlRefresh, IDM_CONTROLREFRESH);
        pDatAnPane = ZNEW DatAnDatAnPane(this, ZNEW zGravSizer(ZGRAV_MIDDLE,0,sizer()));
        pDatAnPane->show();
        sizer()->update();
        // zpb_begin WDatAnConstructor
        pTQ->TQ[pTQ->newTrkAn]->AnWn = this; //this is a track analysis window
        // zpb_end
        show();
    }

WDatAn::~WDatAn() {
    // zpb_begin WDatAnDestructor1
    pTQ->TQ[pDatAnPane->TrkIdx]->AnWn = NULL;
    // zpb_end
}

//
// Menu Item Selection Handlers
//

int WDatAn::cmdControlRefresh(zCommandEvent* ev) {
    DSymRfsh* p=ZNEW DSymRfsh(this, zResId(IDD_SymRfsh));
    p->modal();
    if (p->completed()) {
        // zpb_begin WDatAnControlRefresh
        // zpb_end
    } else {
        // zpb_begin WDatAnControlRefreshCancel
        // zpb_end
    }
    delete p;
    return 0;
}

// zpb_begin WDatAnMemberFunctions
int WDatAn::rtrdrw() {
    zDrawEvt *dummyEv;
    pDatAnPane->draw(dummyEv); //the draw routine does not use this input param
                                // zApp uses this as an event token for its
                                // own dispatching

    return 0;
} //end rtrdrw()

// zpb_end

// Pane Member Functions - jtidsldjpane
jtidsldjpane::jtidsldjpane(zWindow *w, zSizer *sz) : zPane(w, sz, WS_CHILD) {

    // zpb_begin jtidsldjpaneConstructor1
    pjimg = new zBitmap(canvas(), "bground00.bmp");
    // zpb_end
    show();
}

int jtidsldjpane::size(zSizeEvt *ev) {
    setDirty();
    // zpb_begin jtidsldjpaneSize
    // zpb_end
    return zWindow::size(ev);
}

int jtidsldjpane::draw(zDrawEvt* ev) {
    canvas()->lock();
    // zpb_begin jpaneDraw
    canvas()->bitmap(pjimg,zPoint(0,0), SRCCOPY);

```

```

        // zpb_end
        canvas()->unlock();
        return 1;
    }

//
// Frame Member Functions - jtidssld
//
// Window Constructor
Wjtjdssld::Wjtjdssld(zMdiAppFrame *w, const char *title)
: zMdiChildFrame(w,ZNEW zSizer(zDialogUnit(0,0),zDialogUnit(517,412)),WS_CHILD, title) {
    // zpb_begin WjtjdssldConstructor2
    // zpb_end
    deleteOnClose(TRUE);
    menu(ZNEW zMenu(this, zResId(IDM_jtidssldjmenu)));
    menu()->setCommand(this, (CommandProc)&Wjtjdssld::cmdInfoDInfo, IDM_INFODINFO);
    menu()->setCommand(this, (CommandProc)&Wjtjdssld::cmdInfoDIInfo, IDM_INFODIINFO);
    pjpane = ZNEW jtidssldjpane(this, ZNEW zSizer(zDialogUnit(0,0),zDialogUnit(517,412)));
    pjpane->show();
    // zpb_begin WjtjdssldConstructor
    // zpb_end
    show();
}

Wjtjdssld::~Wjtjdssld() {
    // zpb_begin WjtjdssldDestructor1
    // zpb_end
}

//
// Menu Item Selection Handlers
//

int Wjtjdssld::cmdInfoDInfo(zCommandEvt* ev) {
    DDDInfo* p=ZNEW DDDInfo(this, zResId(IDD_DDInfo));
    p->modal();
    if (p->completed()) {
        // zpb_begin WjtjdssldInfoDInfo
        // zpb_end
    } else {
        // zpb_begin WjtjdssldInfoDInfoCancel
        // zpb_end
    }
    delete p;
    return 0;
}

int Wjtjdssld::cmdInfoDIInfo(zCommandEvt* ev) {
    DDDIInfo* p=ZNEW DDDIInfo(this, zResId(IDD_DDIInfo));
    p->modal();
    if (p->completed()) {
        // zpb_begin WjtjdssldInfoDIInfo
        // zpb_end
    } else {
        // zpb_begin WjtjdssldInfoDIInfoCancel
        // zpb_end
    }
    delete p;
    return 0;
}

// zpb_begin WjtjdssldMemberFunctions
int Wjtjdssld::dinfo(DDDInfo *DDptr) {
    //DDDInfo *DDptr;
    DDptr->_mmHgt = pjpane->canvas()->mmHeight();
    DDptr->_mmWth = pjpane->canvas()->mmWidth();
}

```

```

        DDptr->_pxHgt = pjpane->canvas()->pixHeight();
        DDptr->_pxWth = pjpane->canvas()->pixWidth();
        DDptr->_pxPinX = pjpane->canvas()->pixPerInchX();
        DDptr->_pxPinY = pjpane->canvas()->pixPerInchY();
        DDptr->_pfm = pjpane->canvas()->polyFillMode();
        return 0;
    }

    int Wjtdssld::diinfo(DDDIInfo *DDIptr) {
        //DDDIInfo *DDptr;
        zDisplayInfo *zdisptr;
        zdisptr = ZNEW zDisplayInfo(pjpane->canvas());
        int numclr;
        zdisptr->lock();
        numclr = zdisptr->colorRes();
        DDIptr->_aspectX = zdisptr->aspectX();
        DDIptr->_aspectXY = zdisptr->aspectXY();
        DDIptr->_aspectY = zdisptr->aspectY();
        DDIptr->_colorPlanes = zdisptr->colorPlanes();
        DDIptr->_colorRes = zdisptr->colorRes();
        DDIptr->_numcolors = zdisptr->numColors();
        DDIptr->_pixDepth = zdisptr->pixDepth();
        zdisptr->unlock();
        //char tbuf[256];
        //ostream tmpstr(tbuf, 256);
        //tmpstr << numclr << ends;
        //zString sbuf = "          ";
        //sbuf = tbuf;

        //canvas()->pushFont(new zFont("Helv",zPrPoint(12,25,canvas()),900,ffDontCare));
        //canvas()-> text(5,5,sbuf);
        //delete canvas()->popFont();

        return 0;
    }

// zpb_end

//
// Dialog Member Functions - About
//
DAbout::DAbout(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DAboutConstructor2
    // zpb_end
    // zpb_begin DAboutConstructor
    // zpb_end
    centerWindow(this);           // Center window on parent
    show();
}

// zpb_begin DAboutMemberFunctions
// zpb_end

//
// Dialog Member Functions - MsgRfsh
//
DMsgRfsh::DMsgRfsh(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DMsgRfshConstructor2
    // zpb_end
    pDRt = ZNEW zComboBoxStatic(this, ID_DRT, &_DRT);
    pDRt->add(zString(zResId(IDS_MSGRFSHDRT_1)));
    pDRt->add(zString(zResId(IDS_MSGRFSHDRT_2)));
    pDRt->add(zString(zResId(IDS_MSGRFSHDRT_3)));
    pDRt->add(zString(zResId(IDS_MSGRFSHDRT_4)));
    // zpb_begin DMsgRfshConstructor
    WDXWin *tptr;

```

```

    tptr = (WDxWin *) w;
    long tmpval;
    tmpval = (long)tptr->drwRt;
    CurRt(tmpval);
    pDRt->setToDefault();
    // zpb_end
    centerWindow(this);          // Center window on parent
    show();
}

// zpb_begin DMsgRfshMemberFunctions
int DMsgRfsh::CurRt(long CRt){
    char tbuf[256];
    ostrstream tmpstr(tbuf,strlen(tbuf));
    tmpstr << dec << CRt;
    strcpy(_DRt,tbuf);
    return 0;
} //end CurRt()
// zpb_end

//
// Dialog Member Functions - MsgRate
//
DMsgRate::DMsgRate(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DMsgRateConstructor2
    // zpb_end
    pMrt = ZNEW zComboBoxFull(this, ID_MRT, &_Mrt);
    pMrt->add(zString(zResId(IDS_MSGRATMRT_1)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_2)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_3)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_4)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_5)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_6)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_7)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_8)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_9)));
    pMrt->add(zString(zResId(IDS_MSGRATMRT_10)));
    // zpb_begin DMsgRateConstructor
    WDxWin *tptr;
    tptr = (WDxWin *) w;
    long tmpval;
    tmpval = (long)tptr->msgRt;
    CurRt(tmpval);
    pMrt->setToDefault();
    // zpb_end
    centerWindow(this);          // Center window on parent
    show();
}

// zpb_begin DMsgRateMemberFunctions
int DMsgRate::CurRt(long CRt){
    char tbuf[256];
    ostrstream tmpstr(tbuf,strlen(tbuf));
    tmpstr << dec << CRt;
    strcpy(_Mrt,tbuf);
    return 0;
} //end CurRt()
// zpb_end

//
// Dialog Member Functions - TInfo
//
DTInfo::DTInfo(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DTInfoConstructor2
    // following fragment shows how to create and delete this dialog
    //ref only: DTInfo* p=ZNEW DTInfo(this, zResId(IDD_TInfo));
    //ref only: p->modal();

```

```

//ref only: if (p->completed()) {
// zpb_TrkWinTrkPanelButtonDown
//ref only: }
//ref only: delete p;

// zpb_end
pETAlt = ZNEW zEditLine(this, ID_ETALT, &_ETAlt, FLD_NOTREQUIRED);
pETTyp = ZNEW zEditLine(this, ID_ETTYP, &_ETTyp, FLD_NOTREQUIRED);
pETNum = ZNEW zEditLine(this, ID_ETNUM, &_ETNum, FLD_NOTREQUIRED);
pETSpd = ZNEW zEditLine(this, ID_ETSPD, &_ETSpd, FLD_NOTREQUIRED);
pETlocx = ZNEW zEditLine(this, ID_ETLOCX, &_ETlocx, FLD_NOTREQUIRED);
pETlocy = ZNEW zEditLine(this, ID_ETLOCY, &_ETlocy, FLD_NOTREQUIRED);
pDType = ZNEW zComboBoxStatic(this, ID_DTYPE, &_DType);
pDType->add(zString(zResId(IDS_TINFODTYPE_1)));
pDType->add(zString(zResId(IDS_TINFODTYPE_2)));
pDType->add(zString(zResId(IDS_TINFODTYPE_3)));
pDType->add(zString(zResId(IDS_TINFODTYPE_4)));
pDType->add(zString(zResId(IDS_TINFODTYPE_5)));
pDType->add(zString(zResId(IDS_TINFODTYPE_6)));
pDType->add(zString(zResId(IDS_TINFODTYPE_7)));
pDType->add(zString(zResId(IDS_TINFODTYPE_8)));
pDType->add(zString(zResId(IDS_TINFODTYPE_9)));
// zpb_begin DTInfoConstructor

int dnx = pTQ->DTindx;
CrVI(_ETAlt, pTQ->TQ[dnx]->altRaw);
pETAlt->setToDefault();
strcpy(_ETTyp, pTQ->TQ[dnx]->typStr);
pETTyp->setToDefault();
strcpy(_ETNum, pTQ->TQ[dnx]->numStr);
pETNum->setToDefault();
CrVI(_ETlocx, pTQ->TQ[dnx]->xRaw);
pETlocx->setToDefault();
CrVI(_ETlocy, pTQ->TQ[dnx]->yRaw);
pETlocy->setToDefault();
CrVI(_ETSpd, pTQ->TQ[dnx]->spRaw);
pETSpd->setToDefault();
strcpy(_DType, pTQ->TQ[dnx]->dTypStr);
pDType->setToDefault();
// zpb_end
centerWindow(this);           // Center window on parent
show();

}

// zpb_begin DTInfoMemberFunctions
int DTInfo::CrVI(char *cvbuf, float cv){
    char tbuf[256] = " ";
    ostream tmpstr(tbuf, strlen(tbuf));
    tmpstr << cv;
    strcpy(cvbuf, tbuf);
    return 0;
}

//end CrVI()

int DTInfo::UpdAtt(char *typstr){
    int tmptype;
    tmptype = pDType->selection();
    switch(tmptype) {

        case HOSTILE_AIR:
            strcpy(typstr, "HOSTILE AIRCRAFT");
            break;
        case HOSTILE_SUB:
            strcpy(typstr, "HOSTILE SUBSURFACE");
            break;
        case HOSTILE_SURFACE:
            strcpy(typstr, "HOSTILE SURFACE");
            break;
        case UNKNOWN_AIR:
            strcpy(typstr, "UNKNOWN AIRCRAFT");
    }
}

```

```

        break;
    case UNKNOWN_SUB:
        strcpy(typstr, "UNKNOWN SUBSURFACE");
        break;
    case UNKNOWN_SURFACE:
        strcpy(typstr, "UNKNOWN SURFACE");
        break;
    case FRIENDLY_AIR:
        strcpy(typstr, "FRIENDLY AIRCRAFT");
        break;
    case FRIENDLY_SUB:
        strcpy(typstr, "FRIENDLY SUBSURFACE");
        break;
    case FRIENDLY_SURFACE:
        strcpy(typstr, "FRIENDLY SURFACE");
        break;

}; //end case statement

//strcpy(typstr, "HOSTILE AIRCRAFT");
return tmptype;
} //end CrVl()
// zpb_end

//
// Dialog Member Functions - SymRfsh
//
DSymRfsh::DSymRfsh(zWindow *w, const zResId& rid) : zFormDialog(w, rid) {
    // zpb_begin DSymRfshConstructor2
    // zpb_end
    pDRt = ZNEW zComboBoxStatic(this, ID_DRT, &_DRT);
    pDRt->add(zString(zResId(IDS_SYMRFSHDRT_1)));
    pDRt->add(zString(zResId(IDS_SYMRFSHDRT_2)));
    pDRt->add(zString(zResId(IDS_SYMRFSHDRT_3)));
    pDRt->add(zString(zResId(IDS_SYMRFSHDRT_4)));
    // zpb_begin DSymRfshConstructor
    // zpb_end
    centerWindow(this);          // Center window on parent
    show();
}

// zpb_begin DSymRfshMemberFunctions
// zpb_end

//
// Dialog Member Functions - TrkRfsh
//
DTrkRfsh::DTrkRfsh(zWindow *w, const zResId& rid) : zFormDialog(w, rid) {
    // zpb_begin DTrkRfshConstructor2
    // zpb_end
    pDRt = ZNEW zComboBoxStatic(this, ID_DRT, &_DRT);
    pDRt->add(zString(zResId(IDS_TRKRFSHDRT_1)));
    pDRt->add(zString(zResId(IDS_TRKRFSHDRT_2)));
    pDRt->add(zString(zResId(IDS_TRKRFSHDRT_3)));
    pDRt->add(zString(zResId(IDS_TRKRFSHDRT_4)));
    // zpb_begin DTrkRfshConstructor
    WDXWin *tpr;
    tpr = (WDXWin *) w;
    long tmpval;
    tmpval = (long)tpr->drwRt;
    CurRt(tmpval);
    pDRt->setToDefault();
    // zpb_end
    centerWindow(this);          // Center window on parent
    show();
}

// zpb_begin DTrkRfshMemberFunctions

```



```

int DTrkRfsh::CurRt(long CRt){
    char tbuf[256];
    ostrstream tmpstr(tbuf,strlen(tbuf));
    tmpstr << dec << CRt;
    strcpy(_DRt,tbuf);
    return 0;
} //end CurRt()
// zpb_end

//
// Dialog Member Functions - TkSet
//
DTkSet::DTkSet(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DTkSetConstructor2
    // zpb_end
    _laMn = 0;
    _laMx = 0;
    _lgMn = 0;
    _lgMx = 0;
    plaMn = ZNEW zFloatEdit(this, ID_LAMN, &_laMn, "#####.##", FLD_NOTREQUIRED);
    plaMx = ZNEW zFloatEdit(this, ID_LAMX, &_laMx, "#####.##", FLD_NOTREQUIRED);
    plgMn = ZNEW zFloatEdit(this, ID_LGMN, &_lgMn, "#####.##", FLD_NOTREQUIRED);
    plgMx = ZNEW zFloatEdit(this, ID_LGMX, &_lgMx, "#####.##", FLD_NOTREQUIRED);
    // zpb_begin DTkSetConstructor
    _laMn = pTQ->minlt; //this is for op area field
    _laMx = pTQ->maxlt;
    _lgMn = pTQ->minlg;
    _lgMx = pTQ->maxlg;
    plaMn->setToDefault();
    plaMx->setToDefault();
    plgMn->setToDefault();
    plgMx->setToDefault();
    // zpb_end
    centerWindow(this); // Center window on parent
    show();
}

// zpb_begin DTkSetMemberFunctions
// zpb_end

//
// Dialog Member Functions - Rescale
//
DRescale::DRescale(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DRescaleConstructor2
    // zpb_end
    _laMn = 0;
    _laMx = 0;
    _lgMn = 0;
    _lgMx = 0;
    plaMn = ZNEW zFloatEdit(this, ID_LAMN, &_laMn, "#####.##", FLD_NOTREQUIRED);
    plaMx = ZNEW zFloatEdit(this, ID_LAMX, &_laMx, "#####.##", FLD_NOTREQUIRED);
    plgMn = ZNEW zFloatEdit(this, ID_LGMN, &_lgMn, "#####.##", FLD_NOTREQUIRED);
    plgMx = ZNEW zFloatEdit(this, ID_LGMX, &_lgMx, "#####.##", FLD_NOTREQUIRED);
    // zpb_begin DRescaleConstructor
    pTQ->rescale();
    _laMn = pTQ->minlt; //this is for op area field
    _laMx = pTQ->maxlt;
    _lgMn = pTQ->minlg;
    _lgMx = pTQ->maxlg;
    plaMn->setToDefault();
    plaMx->setToDefault();
    plgMn->setToDefault();
    plgMx->setToDefault();
    // zpb_end
    centerWindow(this); // Center window on parent
    show();
}

```

```

}

// zpb_begin DRescaleMemberFunctions
// zpb_end

//
// Dialog Member Functions - FxdFmt
//
DFxdFmt::DFxdFmt(zMDIAppFrame *w,const zResId& rid) : zMDIFormDialog(w,rid) {
    // zpb_begin DFxdFmtConstructor2
    // zpb_end
    // zpb_begin DFxdFmtConstructor
    // zpb_end
    show();
}

// zpb_begin DFxdFmtMemberFunctions
// zpb_end

//
// Dialog Member Functions - DDInfo
//
DDDInfo::DDDInfo(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DDDInfoConstructor2
    Wjtidssld *wptr;
    wptr = ((Wjtidssld *) w);
    // _Edit1 = wptr->pjpane->canvas()->mmHeight();
    int stat;
    stat = wptr->dinfo(this);
    // zpb_end
    _mmHgt = 0;
    _mmWth = 0;
    _pxHgt = 0;
    _pxWth = 0;
    _pxPinX = 0;
    _pxPinY = 0;
    _pfm = 0;
    pmmHgt = ZNEW zIntEdit(this, ID_MMHGT, &_mmHgt, "#####", FLD_NOTREQUIRED);
    pmmWth = ZNEW zIntEdit(this, ID_MMWTH, &_mmWth, "#####", FLD_NOTREQUIRED);
    ppxHgt = ZNEW zIntEdit(this, ID_PXHGT, &_pxHgt, "#####", FLD_NOTREQUIRED);
    ppxWth = ZNEW zIntEdit(this, ID_PXWTH, &_pxWth, "#####", FLD_NOTREQUIRED);
    ppxPinX = ZNEW zIntEdit(this, ID_PXPINX, &_pxPinX, "#####", FLD_NOTREQUIRED);
    ppxPinY = ZNEW zIntEdit(this, ID_PXPINY, &_pxPinY, "#####", FLD_NOTREQUIRED);
    ppfm = ZNEW zIntEdit(this, ID_PFM, &_pfm, "#####", FLD_NOTREQUIRED);
    // zpb_begin DDDInfoConstructor
    // zpb_end
    centerWindow(this);          // Center window on parent
    show();
}

// zpb_begin DDDInfoMemberFunctions
// zpb_end

//
// Dialog Member Functions - DDInfo
//
DDDInfo::DDDInfo(zWindow *w,const zResId& rid) : zFormDialog(w,rid) {
    // zpb_begin DDDInfoConstructor2
    Wjtidssld *wptr;
    wptr = ((Wjtidssld *) w);
    // _Edit1 = wptr->pjpane->canvas()->mmHeight();
    int stat;
    stat = wptr->diinfo(this);
    // zpb_end
    _aspectX = 0;
    _aspectXY = 0;
    _aspectY = 0;

```

```

    _colorPlanes = 0;
    _colorRes = 0;
    _numcolors = 0;
    _pixDepth = 0;
    paspectX = ZNEW zIntEdit(this, ID_ASPECTX, &_aspectX, "#####", FLD_NOTREQUIRED);
    paspectXY = ZNEW zIntEdit(this, ID_ASPECTXY, &_aspectXY, "#####", FLD_NOTREQUIRED);
    paspectY = ZNEW zIntEdit(this, ID_ASPECTY, &_aspectY, "#####", FLD_NOTREQUIRED);
    pcolorPlanes = ZNEW zIntEdit(this, ID_COLORPLANES, &_colorPlanes, "#####", FLD_NOTREQUIRED);
    pcolorRes = ZNEW zIntEdit(this, ID_COLORRES, &_colorRes, "#####", FLD_NOTREQUIRED);
    pnumcolors = ZNEW zIntEdit(this, ID_NUMCOLORS, &_numcolors, "#####", FLD_NOTREQUIRED);
    ppixDepth = ZNEW zIntEdit(this, ID_PIXDEPTH, &_pixDepth, "#####", FLD_NOTREQUIRED);
    // zpb_begin DDDIInfoConstructor
    // zpb_end
    centerWindow(this);           // Center window on parent
    show();
}

// zpb_begin DDDIInfoMemberFunctions
// zpb_end

//
// Simple function to center window within parent or screen
//
void centerWindow(zWindow *w, BOOL fOnParent) {
    zRect winRect, parentRect;
    int xWin, yWin;

    w->getExterior(winRect);
    zSystemInfo screen;

    if (fOnParent && w->parent()) {
        // retrieve parent rectangle
        w->parent()->getExterior(parentRect);

        // center within parent window
        xWin = parentRect.left() + ((parentRect.width() - winRect.width())/2);
        yWin = parentRect.top() + ((parentRect.height() - winRect.height())/2);

        // adjust win x-location for screen size
        if (xWin+winRect.width() > screen.pixWidth() )
            xWin = screen.pixWidth() - winRect.width();

        // adjust win y-location for screen size
        if (yWin+winRect.height() > screen.pixHeight() )
            yWin = screen.pixHeight() - winRect.height();
    }
    else {
        // center within entire screen
        xWin = (screen.pixWidth() - winRect.width()) / 2;
        yWin = (screen.pixHeight() - winRect.height()) / 2;
    }

    // move window to new location
    w->move( (xWin>0) ? xWin : 0,
            (yWin>0) ? yWin : 0,
            winRect.width(),
            winRect.height());
}

//
// Bitmap Pane Member Functions
//
zfBitmapPane::zfBitmapPane(zWindow* w, const zResId &id, zSizer* sz)
    :zPane(w, sz), theBitmap(id) {
    show();
}

```

```

int zfbBitmapPane::draw(zDrawEvt *e) {
    zRect r;
    getInterior(r);
    canvas()->lock();

    zBitmapDisplay *bd;
    bd = ZNEW zBitmapDisplay(&theBitmap);
    bd->lock();
    bd->copyTo(canvas(), r.left(), r.top(),
theBitmap.size().width(), theBitmap.size().height(), 0, 0);
    bd->setBitmap(0);
    bd->unlock();
    delete bd;
    canvas()->unlock();
    return TRUE;
}

// zpb_begin AppUserCode
// zpb_end

//
// Application Entry Point
//
void zApp::main() {
    initIntPack();
    // zpb_begin AppMain
    // zpb_end
    WMain* p=ZNEW WMain(zString(zResId(IDS_MAIN)));
    // zpb_begin AppMain2
    // zpb_end
    go();
    delete p;

    // zpb_begin AppMain3
    // zpb_end
}

/*****

#include "output.h"
#include "..\include\output.h"
#include <time.h>

/*****
int InitOutput(HINSTANCE hInstance){

    WNDCLASS wndclass;

    //check to see if the class is already registered
    if(GetClassInfo(hInstance,"Output",&wndclass))
        return TRUE;

    //register the class
    wndclass.style
    wndclass.lpfnWndProc
    wndclass.cbClsExtra
    wndclass.cbWndExtra
    wndclass.hInstance
    wndclass.hIcon
    wndclass.hCursor
    wndclass.hbrBackground
    wndclass.lpszMenuName
    wndclass.lpszClassName

    = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    = OutputWndProc;
    = 10 ;
    = 10 ;
    = hInstance ;
    = LoadIcon(NULL, IDI_APPLICATION);
    = LoadCursor(NULL, IDC_ARROW);
    = NULL;
    = NULL;
    = "Output";

```

```

    RegisterClass(&wndclass);

    //make sure the class is registered
    if(GetClassInfo(hInstance,"Output",&wndclass))
        return TRUE;

    return FALSE;
}

/*****
OutputWndProc
    Purpose
    Window Message Proc
*****/
long far pascal OutputWndProc(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam){

    int                tx,y;
    int                oldnumlines;
    int                position;
    int                len;
    int                update;
    char *             c;
    SIZE               size;
    RECT               rect;
    HDC                hdc;

    OUTPUT *           op;
    OUTPUTITEM * itemlist;

    //*****
    //if this is the first message then setup the data structure
    if(message == WM_NCCREATE){

        //create a new output structure and store it
        op = new OUTPUT;
        SetWindowLong(hwnd,0,(long)op);

        //set up the initial variable values
        op->font                = NULL;
        op->numlines            = 0;
        op->numscreenlines      = 0;
        op->numitemsloaded      = 0;
        op->viewoffset          = 0;
        op->scrollrange         = 0;
        op->margin              = 2;
        op->textcolor            = GetSysColor(COLOR_WINDOWTEXT);
        op->backcolor           = GetSysColor(COLOR_WINDOW);
        op->textalign            = 1; //left
        op->enablelog            = FALSE;
        op->datedlog            = FALSE;
        op->filename            = NULL;
        op->fptr                = NULL;
        op->historysize         = 0;
        op->items               = NULL;
        op->lineheight          = 0;
        op->lastrowclicked      = -1;

        op->hscrollrange        = 0;
        op->maxlinewidth        = 0;
        op->hviewoffset         = 0;

        op->stamptype           = 0;
    }
}

```

```

//*****
//get the pointer to the data structure
else{
op = (OUTPUT *)GetWindowLong(hwnd,0);
}

//*****
//if the datastructure is NULL then just use the default message handling
if(op==NULL){
return DefWindowProc(hwnd,message,wParam,lParam);
}

//*****
//**** process messages ****
switch(message){

//*****
case WM_CREATE:{
SendMessage(hwnd,WM_SIZE,0,0);
return 1;
}

//*****
// if destroying then delete all data
case WM_NCDESTROY:{

if(op->items != NULL){
for( t = 0 ; t < op->numlines ; t++ ){
if(op->items[t].string != NULL)
delete[] op->items[t].string;
}
delete[] op->items;
}

if(op->filename != NULL)
delete[] op->filename;

if(op->fptr != NULL)
fclose(op->fptr);

delete op;
op = NULL;

SetWindowLong(hwnd,0,(long)op);
return 1;
}

//*****
// if sizing then get the new number of lines
// then allocate or delete memory for the new number of lines
case WM_SIZE:{

//get the height of a character
hdc = GetDC(hwnd);
if(op->font !=NULL)
SelectObject(hdc,op->font);
GetTextExtentPoint(hdc,"X",1,&size);
op->lineheight = size.cy;
ReleaseDC(hwnd,hdc);

GetClientRect(hwnd,&op->clientrect);

//get the number of total rows and the number that can be displayed
oldnumlines = op->numlines; //old num rows

```

```

rows      op->numlines = (op->clientrect.bottom / op->lineheight) + 1 + op->historysize; //new num

rows      if(op->numlines < op->historysize) //double check
           op->numlines = op->historysize;
           op->numscreenlines = (op->clientrect.bottom / op->lineheight) + 1; //number of visible

rows      if(op->numscreenlines < 1) //double check
           op->numlines = 1;

           //adjust the view offset
           if(op->numlines <= op->numscreenlines)
               op->viewoffset=0;

           //realloc the mem
           itemlist = new OUTPUTITEM[op->numlines];

           //copy the info from the old list to the new one
           for(t=0; t < op->numlines;t++){
               if(t < oldnumlines){
                   itemlist[t].string = op->items[t].string;
                   itemlist[t].color = op->items[t].color;
                   itemlist[t].align = op->items[t].align;
                   itemlist[t].width = op->items[t].width;
               }
               else{
                   itemlist[t].string = NULL;
                   itemlist[t].width = 0;
               }
           }

           //remove the old info
           for(t=op->numlines;t<oldnumlines;t++){
               if(op->items[t].string != NULL)
                   delete[] op->items[t].string;
               op->items[t].width = 0;
           }
           delete[] op->items;

           //reset the pointer
           op->items = itemlist;

           //adjust the number of loaded items
           if(op->numitemsloaded > op->numlines)
               op->numitemsloaded = op->numlines;

           //adjust the scrollbars
           AdjustScrollBars(hwnd,op);

           return 1;
       }

//*****
// paint the window
case WM_PAINT:{

           //set up the device context
           hdc = GetDC(hwnd);
           if(op->font !=NULL)
               SelectObject(hdc,op->font);
           SetBkColor(hdc,op->backcolor);

           //get the client area
           GetClientRect(hwnd,&rect);
           y= rect.bottom;

           if( op->numitemsloaded >= op->numscreenlines){

```

```

x= op->viewoffset/op->lineheight;
y = op->viewoffset%op->lineheight;
for( t = rect.bottom+y ; t > 0 ; t= op->lineheight){

    //set up the rectangle
    rect.bottom = t;
    rect.top = rect.bottom - op->lineheight;
    //setup the alignment
    if(op->items[x].align ==1){ //left
        SetTextAlign(hdc,TA_LEFT);
        position = rect.left + op->margin;
    }
    else if(op->items[x].align ==2){ //center
        SetTextAlign(hdc,TA_CENTER);
        position = (rect.right - rect.left)/2;
    }
    else if(op->items[x].align ==3){ //right
        SetTextAlign(hdc,TA_RIGHT);
        position = rect.right - op->margin;
    }
    //set the text color
    SetTextColor(hdc,op->items[x].color);
    //draw the line
    ExtTextOut(hdc,position - op->hviewoffset,rect.top,ETO_OPAQUE,&rect,
        op->items[x].string,lstrlen(op->items[x].string),NULL);

    x++;
}
}
else{
    for(t= op->numitemsloaded -1;t >=0;t--){
        //set up the rect
        rect.bottom = rect.top + op->lineheight;
        //setup the alignment
        if(op->items[t].align ==1){ //left
            SetTextAlign(hdc,TA_LEFT);
            position = rect.left + op->margin;
        }
        else if(op->items[t].align ==2){ //center
            SetTextAlign(hdc,TA_CENTER);
            position = (rect.right - rect.left)/2;
        }
        else if(op->items[t].align ==3){ //right
            SetTextAlign(hdc,TA_RIGHT);
            position = rect.right - op->margin;
        }
        //set the text color
        SetTextColor(hdc,op->items[t].color);
        //draw the text
        ExtTextOut(hdc,position- op->hviewoffset- op-
>hviewoffset,rect.top,ETO_OPAQUE,&rect,
            op->items[t].string,lstrlen(op->items[t].string),NULL);

        rect.top = rect.bottom;
    }
    rect.bottom = y;
    ExtTextOut(hdc,rect.left,rect.top,ETO_OPAQUE,&rect,"",0,NULL);
}

ReleaseDC(hwnd,hdc);
ValidateRect(hwnd,NULL);
return 1;
}
//*****
//mouse clicks
case WM_LBUTTONDOWN:
case WM_RBUTTONDOWN:

```



```

case WM_LBUTTONDOWN:{
    //find the row that was clicked in
    if( op->numitemsloaded >= op->numscreenlines){
        y = op->clientrect.bottom - (short)HIWORD(lParam) + op->viewoffset;
        op->lastrowclicked = y / op->lineheight;
    }
    else{
        y = (short)HIWORD(lParam);
        op->lastrowclicked = op->numitemsloaded - (y / op->lineheight) - 1;
    }
    if(message == WM_LBUTTONDOWN)
        SendNotifyMessage(hwnd,OPN_LCLICKED);
    else if(message == WM_RBUTTONDOWN)
        SendNotifyMessage(hwnd,OPN_RCLICKED);
    else if(message == WM_LBUTTONDOWNBLCLK)
        SendNotifyMessage(hwnd,OPN_DCLICKED);
    return 0;
}

}

//*****
//set the new font then recalc the number of lines
case WM_SETFONT:{
    op->font = (HFONT)wParam;
    SendMessage(hwnd,WM_SIZE,0,0);
    return 1;
}

//*****
//vertical scroll bar
case WM_VSCROLL:{
    y = op->viewoffset;

    switch(LOWORD(wParam)){
        case SB_BOTTOM :{
            op->viewoffset = 0;
            break;
        }
        case SB_LINEDOWN:{
            op->viewoffset -= op->lineheight;
            break;
        }
        case SB_LINEUP:{
            op->viewoffset += op->lineheight;
            break;
        }
        case SB_PAGEDOWN:{
            op->viewoffset = op->clientrect.bottom;
            break;
        }
        case SB_PAGEUP:{
            op->viewoffset += op->clientrect.bottom;
            break;
        }
        case SB_THUMBTRACK:
        case SB_THUMBPOSITION:{
            #ifdef WIN32
                op->viewoffset = op->scrollrange - HIWORD(wParam);
            #else
                op->viewoffset = op->scrollrange - LOWORD(lParam);
            #endif
            break;
        }
        case SB_TOP:{
            op->viewoffset = op->scrollrange;
            break;
        }
    }
}
}

```

```

        //check the range
        if(op->viewoffset < 0)
            op->viewoffset = 0;
        if(op->viewoffset > op->scrollrange)
            op->viewoffset = op->scrollrange;
        //check for change in position
        if(y != op->viewoffset){
            //update
            SetScrollPos(hwnd,SB_VERT,op->scrollrange - op->viewoffset,TRUE);
            InvalidateRect(hwnd,NULL,TRUE);
        }
        return 0;
    }
}
//*****
//vertical scroll bar
case WM_HSCROLL:{

    x = op->hviewoffset;

    switch(LOWORD(wParam)){
        case SB_BOTTOM:{
            op->hviewoffset = op->scrollrange;;
            break;
        }
        case SB_LINEDOWN:{
            op->hviewoffset += 10;
            break;
        }
        case SB_LINEUP:{
            op->hviewoffset -= 10;
            break;
        }
        case SB_PAGEDOWN:{
            op->hviewoffset += op->clientrect.right;
            break;
        }
        case SB_PAGEUP:{
            op->hviewoffset -= op->clientrect.right;
            break;
        }
        case SB_THUMBPOSITION:
        case SB_THUMBTRACK:{
            #ifdef WIN32
                op->hviewoffset = HIWORD(wParam);
            #else
                op->hviewoffset = LOWORD(lParam);
            #endif
            break;
        }
        case SB_TOP:{
            op->hviewoffset = 0;
            break;
        }
    }

    //check the range
    if(op->hviewoffset < 0)
        op->hviewoffset = 0;
    if(op->hviewoffset > op->hscrollrange)
        op->hviewoffset = op->hscrollrange;

    //check for change in position
    if(x != op->hviewoffset){
        //update
        SetScrollPos(hwnd,SB_HORZ,op->hviewoffset,TRUE);
        InvalidateRect(hwnd,NULL,TRUE);
    }
    return 0;
}

```

```

    }
//*****
//add a new line of text
case OP_ADDLINE:{

    //params
    c          = (char *)lParam;
    position = wParam;
    if(position < 0)
        position = 0;

    //check to see if the line being remove was the widest
    //if so then find the new widest item
    x = op->numlines - 1;
    if(op->items[x].width == op->maxlinewidth){
        op->maxlinewidth = 0;
        op->items[x].width = 0;
        for(t=0;t< op->numlines;t++){
            if(op->items[t].width > op->maxlinewidth)
                op->maxlinewidth = op->items[t].width;
        }
    }
    //remove the last string - at the very end of the list
    if(op->items[x].string != NULL)
        delete[] op->items[x].string;

    //move all the strings color and alignment info above the
    //specified entry position
    for(t=(op->numlines-1);t>position;t-){
        op->items[t].string = op->items[t-1].string;
        op->items[t].color = op->items[t-1].color;
        op->items[t].align = op->items[t-1].align;
        op->items[t].width = op->items[t-1].width;
    }

    //add the new string color and alignment info
    if(c == NULL)
        c = "";
    //copy the string and expand tabs
    op->items[position].string = ExpandTabs(c );

    op->items[position].color = op->textcolor;
    op->items[position].align = op->textalign;

    //get the width of the string
    hdc = GetDC(hwnd);
    if(op->font != NULL)
        SelectObject(hdc,op->font);
    GetTextExtentPoint(hdc,c,strlen(c),&size);
    op->items[position].width =size.cx;
    ReleaseDC(hwnd,hdc);

    //check to see if this is a max width
    if(op->maxlinewidth < size.cx )
        op->maxlinewidth = size.cx;

    //adjust the number of items loaded
    op->numitemsloaded++;
    if(op->numitemsloaded > op->numlines)
        op->numitemsloaded = op->numlines;

    //if logging then add the line to the log
    if(op->enablelog){
        WriteToLog(op,&op->items[position]);
    }
}

```

```

        //adjust the scrollbars
        if(op->viewoffset > 0)
            op->viewoffset += op->lineheight;
        AdjustScrollBars(hwnd,op);

        //re-draw
        InvalidateRect(hwnd,NULL,TRUE);
        return 1;
    }
}
//*****
case OP_ADDSTAMPEDLINE:{
    len = strlen((LPSTR)lParam) + 30;
    c = new char[len];

    GetTimeDateStamp(c,20,op->stamptype);
    lstrcat(c," ");
    lstrcat(c,(LPSTR)lParam);
    SendMessage(hwnd,OP_ADDDLN,wParam,(LPARAM)c);
    delete[] c;

    return 1;
}
//*****
case OP_SETMARGINS:{
    //check for a valid range
    if((int)wParam >=0 && wParam <1000){
        op->margin = wParam;
        //re-draw
        InvalidateRect(hwnd,NULL,TRUE);
        return TRUE;
    }
    return FALSE;
}
//*****
case OP_SETTEXTALIGN:{
    //check for a valid range
    if(wParam >0 && wParam <4){
        op->textalign = wParam;
        //re-draw
        InvalidateRect(hwnd,NULL,TRUE);
        return TRUE;
    }
    else{
        op->textalign = 1;
    }
    return FALSE;
}
//*****
case OP_CLEAR:{
    //delete all strings
    if(op->items != NULL){
        for(t=0;t<op->numlines;t++){
            if(op->items[t].string!=NULL){
                delete[] op->items[t].string;
                op->items[t].string = NULL;
            }
        }
    }

    op->items[t].width = 0;

    op->numitemsloaded = 0;
    op->viewoffset = 0;
    SetScrollRange(hwnd,SB_VERT,0,0,TRUE);
    op->maxlinewidth = 0;
    op->hviewoffset = 0;
    SetScrollRange(hwnd,SB_HORZ,0,0,TRUE);

```

```

        //re-draw
        InvalidateRect(hwnd,NULL,TRUE);
        return 1;
    }
}
//*****
case OP_SETTEXTCOLOR:{
    //set the color
    op->textcolor = (COLORREF)lParam;
    //re-draw
    InvalidateRect(hwnd,NULL,TRUE);
    return 1;
}
//*****
case OP_SETBACKCOLOR:{
    //set the color
    op->backcolor = (COLORREF)lParam;
    //re-draw
    InvalidateRect(hwnd,NULL,TRUE);
    return 1;
}
//*****
case OP_SETLOGNAME:{
    if(op->filename != NULL)
        delete[] op->filename;

    op->filename = new char[strlen((LPCSTR)lParam)+1];
    strcpy(op->filename,(LPCSTR)lParam);

    //if logging is enabled then open the log
    OpenLogFile(op);
    return 1;
}
//*****
case OP_ENABLELOG:{
    if(wParam ==0)
        op->enablelog = FALSE;
    else{
        op->enablelog = TRUE;
        //if logging is enabled then open the log
        OpenLogFile(op);
    }
    return op->enablelog;
}
//*****
case OP_DATEDLOGGING:{
    if(wParam ==0)
        op->datedlog = FALSE;
    else{
        op->datedlog = TRUE;
        //if logging is enabled then open the log
        OpenLogFile(op);
    }
    return op->datedlog;
}
//*****
case OP_HISTORYSIZE:{
    //set the history size
    op->historysize = wParam;
    if(op->historysize < 0)
        op->historysize =0;
    if(op->historysize > 1000)
        op->historysize = 1000;
    //call WM_SIZE to readjust
    SendMessage(hwnd,WM_SIZE,0,0);
    return op->historysize;
}
}

```

```

//*****
case OP_GETROWCLICKED:{
    return op->lastrowclicked;
}
//*****
case OP_GETTEXTLENGTH:{
    if((int)wParam >=0 && (int)wParam < op->numscreenlines){
        return strlen(op->items[wParam].string);
    }
    else{
        return 0;
    }
}
//*****
case OP_GETTEXT:{
    if((int)wParam >=0 && (int)wParam < op->numscreenlines){
        strcpy((LPSTR)lParam,op->items[wParam].string);
        return TRUE;
    }
    return FALSE;
}
//*****
case OP_DELETELINE:{
    if((int)wParam >=0 && (int)wParam < op->numitemsloaded){

        //check to see if the line being removed was the widest
        //if so then find the new widest item
        if(op->items[wParam].width == op->maxlinewidth){
            op->maxlinewidth =0;
            op->items[wParam].width =0;
            for(t=0;t< op->numlines;t++){
                if(op->items[t].width > op->maxlinewidth)
                    op->maxlinewidth = op->items[t].width;
            }
        }

        //delete the string
        if(op->items[wParam].string != NULL)
            delete[] op->items[wParam].string;

        //shift items down one
        for(t=wParam ; t< (op->numitemsloaded -1); t++){
            op->items[t].string = op->items[t+1].string;
            op->items[t].color = op->items[t+1].color;
            op->items[t].align = op->items[t+1].align;
            op->items[t].width = op->items[t+1].width;
        }
        op->items[t].string = NULL;
        op->items[t].width = 0;

        op->numitemsloaded --;

        //adjust the scrollbars
        AdjustScrollBars(hwnd,op);

        //re-draw
        InvalidateRect(hwnd,NULL,TRUE);

        return TRUE;
    }
    return FALSE;
}
//*****
case OP_UPDATELINE:{

    //check the range
    if((int)wParam <0 || (int)wParam >= op->numitemsloaded){

```

```

        return FALSE;
    }

    //get the params
    c = (char *)lParam;
    position = wParam;

    //check to see if the line being updated was the widest
    //if so then find the new widest item
    if(op->items[position].width == op->maxlinewidth)
        update = TRUE;
    else
        update = FALSE;

    //add in extra lines if ness.
    if(position >= op->numitemsloaded){
        for(t = op->numitemsloaded; t < position; t++){
            op->items[t].string = new char[2];
            strcpy(op->items[t].string, "");
            op->items[t].color = op->textcolor;
            op->items[t].align = op->textalign;
            op->items[t].width = 0;
        }
        op->numitemsloaded = position + 1;
    }

    //update the specified line
    if (op->items[position].string != NULL)
        delete[] op->items[position].string;
    op->items[position].string = new char[strlen(c)+1];
    strcpy(op->items[position].string, c);
    op->items[position].color = op->textcolor;
    op->items[position].align = op->textalign;
    op->items[position].width = 0;

    //if update is true then find the widest line
    if(update){
        op->maxlinewidth = 0;
        op->items[wParam].width = 0;
        for(t=0; t < op->numlines; t++){
            if(op->items[t].width > op->maxlinewidth)
                op->maxlinewidth = op->items[t].width;
        }
    }

    //re-draw
    InvalidateRect(hwnd, NULL, TRUE);

    return TRUE;
}

//*****
case OP_STAMPSTYLE:{
    if(wParam == FALSE)
        op->stamptype = FALSE;
    else
        op->stamptype = TRUE;
    return op->stamptype;
}

//*****
case OP_GETNUMLINES:{
    return op->numitemsloaded;
}

//*****
case OP_UPDATESTAMPEDLINE:{
    len = strlen((LPCTSTR)lParam) + 30;
    c = new char[len];

```

```

        GetTimeStamp(c,20,op->stamptype);
        lstrcat(c," ");
        lstrcat(c,(LPSTR)lParam);
        SendMessage(hwnd,OP_UPDATELINE,wParam,(LPARAM)c);
        delete[] c;

        return 1;
    }

}

return DefWindowProc(hwnd,message,wParam,lParam);
}
/*****
*****/
int OpenLogFile(OUTPUT * op){

    char *path;
    int extension,len;
    char date[10];

    //check to see if logging is enabled
    if(op->enablelog == FALSE){
        return FALSE;
    }

    //alloc a string for the filepath
    path = new char[strlen(op->filename)+10];

    //close old log file if open
    if(op->fptr != NULL)
        fclose(op->fptr);

    //get the logfile name if dated logfiles are used
    if(op->datedlog){

        //get the date
        GetDateString(op,date,10);

        //find where the filename extension is
        len = strlen(op->filename);
        for(extension =0;extension < len ;extension++){
            if(op->filename[extension]=='.')
                break;
        }
        if(extension != len){
            op->filename[extension]=0;
            wsprintf(path,"%s%s.%s",op->filename,date,&op->filename[extension+1]);
            op->filename[extension]='.';
        }
        else{
            wsprintf(path,"%s%s",op->filename,date);
        }
    }
    else{
        lstrcpy(path,op->filename);
    }

    //open the log file
    op->fptr = fopen(path,"a+");

    delete[] path;

    if(op->fptr != NULL)
        return TRUE;

    return FALSE;
}

```



```

/*****
*****/
int WriteToLog(OUTPUT *op,OUTPUTITEM *item){

    time_t          tt;
    struct tm *      systime;

    //check the date if datedlogging
    if(op->datedlog){
        //get the time/date
        tt = time(NULL);
        systime = localtime(&tt);
        //store the current date
        if(op->logday != systime->tm_mday || op->logmon != systime->tm_mon ||
           op->logyear != systime->tm_year){
                OpenLogFile(op);
            }
        }
        //write the line
        fwrite(item->string,sizeof(char),strlen(item->string),op->fptr);
        fwrite("\n",sizeof(char),1,op->fptr);

        return TRUE;
    }
/*****
*****/
int GetDateString(OUTPUT *op,LPSTR string,int len){

    time_t          t;
    struct tm *systime;

    if(len < 7)
        return FALSE;

    //get the time/date
    t = time(NULL);
    systime = localtime(&t);

    //store the current date
    op->logday = systime->tm_mday;
    op->logmon = systime->tm_mon;
    op->logyear = systime->tm_year;

    wsprintf(string,"%2.2d%2.2d%2.2d",systime->tm_year,systime->tm_mon+1,
             systime->tm_mday);

    return TRUE;
}
/*****
*****/
type 0 - date + time
1 - time only
*****/
int GetTimeStamp(LPSTR string,int len,int type){

    time_t          t;
    struct tm *systime;

    if(len < 18)
        return FALSE;

    //get the time/date
    t = time(NULL);
    systime = localtime(&t);

    if(type == 0){
        wsprintf(string,"%2.2d%2.2d%2.2d %2.2d:%2.2d:%2.2d",

```

```

        systime->tm_mday,systime->tm_mon+1,systime->tm_year,
        systime->tm_hour,systime->tm_min,systime->tm_sec);
    }
    else if(type == 1){
        wprintf(string,"%2.2d:%2.2d:%2.2d",
        systime->tm_hour,systime->tm_min,systime->tm_sec);
    }
    return TRUE;
}

/*****
*****/
long SendNotifyMessage(HWND hwnd,int message){

#ifdef _WIN32
    long ID = GetWindowLong(hwnd,GWL_ID);
    return SendMessage(GetParent(hwnd),WM_COMMAND,MAKELPARAM(ID,message),(LPARAM)hwnd);
#else
    WORD ID = GetWindowWord(hwnd,GWW_ID);
    return SendMessage(GetParent(hwnd),WM_COMMAND,ID,MAKELPARAM(hwnd,message));
#endif
}

/*****
*****/
int AdjustScrollBars(HWND hwnd,OUTPUT *op){

    //adjust the vertical scroll bar
    if((op->numitemsloaded * op->lineheight) > op->clientrect.bottom ){
        op->scrollrange = (op->numitemsloaded * op->lineheight) - op->clientrect.bottom - 1;
        if(op->viewoffset > op->scrollrange)
            op->viewoffset = op->scrollrange;
        SetScrollRange(hwnd,SB_VERT,0,op->scrollrange,FALSE);
        SetScrollPos(hwnd,SB_VERT,op->scrollrange - op->viewoffset,TRUE);
    }
    else{
        SetScrollRange(hwnd,SB_VERT,0,0,TRUE);
        op->viewoffset = 0;
    }

    //adjust the horizontal scrollbar
    if((op->maxlinewidth + op->margin*2) > op->clientrect.right){
        op->hscrollrange = (op->maxlinewidth + op->margin*2) - op->clientrect.right;
        if(op->hviewoffset > op->hscrollrange)
            op->hviewoffset = op->hscrollrange;
        SetScrollRange(hwnd,SB_HORZ,0,op->hscrollrange,TRUE);
        SetScrollPos(hwnd,SB_HORZ,op->hviewoffset,TRUE);
    }
    else{
        SetScrollRange(hwnd,SB_HORZ,0,0,TRUE);
        op->hviewoffset = 0;
    }

    return TRUE;
}

/*****
*****/
LPSTR ExpandTabs(LPSTR in){

    int t;
    int num = 0;
    int len = strlen(in);
    int pos;

    LPSTR out;

    for(t=0;t<len;t++){
        if(in[t]==9)

            num++;

```

```

    }

    num = (num*5)+len;

    out = new char[num+1];

    pos =0;
    for(t=0;t<len;t++){
        if(in[t]==9){

            out[pos]=32;
            pos++;
            out[pos]=32;
            pos++;
            out[pos]=32;
            pos++;
            out[pos]=32;
            pos++;
            out[pos]=32;
            pos++;

        }
        else{

            out[pos]=in[t];
            pos++;

        }
    }
    out[pos]=0;

    return out;
}

```

```

/*****

```

```

#include <windows.h>
#include <stdio.h>
#include <io.h>
#include <winsock.h>
#include "routines.h"

```

```

/*****
*****/

```

```

long GetFileTime(LPCSTR filename,FTIME * ft){

```

```

    FILE *fptr;
    int rt;

```

```

    /* create a file containing 10 bytes */
    fptr = fopen(filename,"r");

```

```

    rt = getftime(fileno(fptr),(ftime*)ft);

```

```

    /* close the file */
    fclose(fptr);

```

```

    if(rt ==0)
        return TRUE; //success

```

```

    return FALSE;

```

```

}
/*****
*****/

```

```

long GetFileSize(LPCSTR filename){

```

```

    FILE *fptr;
    long size;

```

```

    /* create a file containing 10 bytes */

```

```

    fptr = fopen(filename,"r");

    size = filelength(fileno(fptr));

    /* close the file */
    fclose(fptr);

    return size;
}

/*****
*****/
int FixFilename(LPSTR filename){

    int t,x;
    int len = strlen(filename);

    for(t=0;t<len;t++){
        if(filename[t]=='/'){
            filename[t]='\\';
        }
    }
    //check for two in a row
    for(t=0;t<len;t++){
        if(filename[t]=='\\'){
            if(filename[t+1]=='\\'){
                for(x=t;x<len;x++){
                    filename[x]=filename[x+1];
                }
                len--;
            }
        }
    }

    return TRUE;
}

/*****
return
0 - success
1 - address string is not long enough
2 - failed
*****/
int GetClientAddressFromName(LPSTR address,int maxaddrlen,LPCSTR name){

    hostent *pHostent;
    int len;

    pHostent = gethostbyname (name);

    if(pHostent==NULL)
        return 2;

    len = strlen(pHostent->h_addr_list[0]);
    if(len > (maxaddrlen-1))
        return 1;

    strcpy(address,pHostent->h_addr_list[0]);

    return 0;
}

/*****
0 - success
1 - address string is not long enough
2 - failed
*****/

```

```

int GetClientNameFromAddress(LPSTR name,int maxnamelen,LPCSTR address){

    unsigned long addr;
    hostent *pHostent;
    int len;

    addr = inet_addr (address);

    pHostent = gethostbyaddr((char*)&addr,4,PF_INET);

    if(pHostent==NULL)
        return 2;

    len = strlen(pHostent->h_name);

    if(len > (maxnamelen-1))
        return 1;

    strcpy(name,pHostent->h_name);

    return 0;

}

```

/*****
 SAMPLE DATA FILE USED IN DEMONSTRATION IS CALSSIFIED AND
 CAN NOT BE INCLUDED HERE

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Road, Suite 0944
Fort Belvoir, VA 22060-6218
2. Dudley Knox Library 2
Naval Postgraduate School
Monterey, CA 93943
3. Center for Naval Analysis 1
4401 Ford Ave.
Alexandria, VA 22302
4. Dr. Ted Lewis, Chairman, Code CS/L 1
Computer Science Dept.
Naval Postgraduate School
Monterey, CA 93943
5. Chief of Naval Research 1
800 North Quincy St.
Arlington, VA 22217
6. Dr. Luqi, Code CS/Lq 1
Computer Science Dept.
Naval Postgraduate School
Monterey, CA 93943
7. Dr. Marvin Langston 1
1225 Jefferson Davis Highway
Crystal Gateway 2 / Suite 1500
Arlington, VA 22202-4311
8. David Hislop 1
U.S. Army Research Office
PO Box 12211
Research Triangle Park, NC 27709-2211
9. Capt. Talbot Manvel 1
Naval Sea Systems Command
2531 Jefferson Davis Hwy.
Attn: TMS 378 Capt. Manvel
Arlington, VA 22240-5150

10. CDR Michael McMahon 1
Naval Sea System Command
2531 Jefferson Davis Hwy.
Arlington, VA 22242-5160
11. Dr. Elizabeth Wald1
Office of Naval Research
800 N. Quincy St.
ONR CODE 311
Arlington, VA 22217-5660
12. Dr. Ralph Wachter1
Office of Naval Research
800 N. Quincy St.
CODE 311
Arlington, VA 22217-5660
13. Army Research Lab1
115 O'Keefe Building
Attn: Mark Kendall
Atlanta, GA 30332-0862
14. National Science Foundation1
Attn: Bruce Barnes
Div. Computer & Computation Research
1800 G St. NW
Washington, DC 20550
15. National Science Foundation1
Attn: Bill Agresty
4201 Wilson Blvd.
Arlington, VA 22230
16. Hon. John W. Douglass1
Assistant Secretary of the Navy
(Research, Development and Acquisition)
Room E741
1000 Navy Pentagon
Washington, DC 20350-1000

- 17. Technical Library Branch 1
Naval Command, Control, and Ocean Surveillance Center
RDT&E Division, Code D0724
San Diego, CA 92152-5001
- 18. Head, Command and Control Department1
Naval Command, Control and Ocean Surveillance Center
RDT&E Division, Code D40
San Diego, CA 92152-5001
- 19. Head, Integration and Interoperability Division1
Naval Command, Control and Ocean Surveillance Center
RDT&E Division, Code D45
San Diego, CA 92152-5001
- 20. Michael W DaBose, Technology Development and Insertion Group1
Naval Command, Control and Ocean Surveillance Center
RDT&E Division, Code D4525
San Diego, CA 92152-5001